(Submitted to LICS 2016)

Open Call-by-Value

Beniamino Accattoli
INRIA, UMR 7161, LIX, École Polytechnique
beniamino.accattoli@inria.fr

Giulio Guerrieri
Aix-Marseille Université, Marseille, I2M UMR 7373
giulio.guerrieri@univ-amu.fr, gguerrieri@uniroma3.it

Abstract

The elegant theory of the call-by-value lambda-calculus relies on weak evaluation and closed terms, that are natural hypotheses in the study of programming languages. To model proof assistants, however, strong evaluation and open terms are required, and it is well known that the operational semantics of call-by-value becomes problematic in this case, as first pointed out by Paolini and Ronchi della Rocca. Here we study the intermediate setting—that we call Open Call-by-Value—of weak evaluation with open terms, on top of which Grégoire and Leroy designed the abstract machine of Coq. Various calculi for Open Call-by-Value already exist, coming from logical, semantical, or implementative points of view, each one with its pros and cons. This paper presents a detailed comparative study of their operational semantics. First, we show that all calculi are equivalent from a termination point of view, justifying the slogan Open Call-by-Value. Second, we compare their equational theories. Third, we present a detailed quantitative analysis of the time cost model. Four, we introduce a new simple abstract machine, and prove it a reasonable implementation of Open Call-by-Value with respect to its cost model. Along the way, there emerges a sharp deconstruction of call-by-value evaluation and of the complexity of its implementations.

1. Introduction

Plotkin's call-by-value λ-calculus [28] is at the heart of programming languages such as Ocaml and proof assistants such as Coq. In the study of programming languages, call-by-value (CBV) evaluation is usually *weak*, *i.e.* not under abstractions, and terms are assumed to be *closed*. These constraints give rise to a beautiful theory—let us call it *Closed CBV*—having the following *harmony property*, that relates rewriting and normal forms:

Closed normal forms are values (and values are normal forms)

Harmony expresses a form of internal completeness with respect to unconstrained β-reduction: the restriction to CBV β-reduction (referred to as β_v*-reduction*) has an impact on the order in which redexes are evaluated, but evaluation never gets stuck, as every β-redex will eventually become a β_v-redex and be fired (unless evaluation diverges).

It often happens, however, that one needs to go beyond the perfect setting of Closed CBV, by considering *Strong CBV*, where reduction under abstractions is allowed, or the intermediate setting of *Open CBV*, where evaluation is weak but terms are not necessarily closed. The need arises, most notably, when trying to describe the implementation model of Coq [15], but also from other motivations, as denotational semantics [5, 12, 25, 30], monad and CPS translations and the associated equational theories [14, 18, 23, 31, 32], bisimulations [20], partial evaluation [19], linear logic proof nets [2], or cost models [7].

Naïve Open CBV In call-by-name (CBN) turning to open terms or strong evaluation is harmless because CBN does not impose any special form to the arguments of β-redexes. On the contrary, turning to Open or Strong CBV is delicate. If one simply considers Plotkin's weak β_v-reduction on open terms—let us call it *Naïve Open CBV*—then harmony does no longer hold, as there are open β-normal forms that are not values (*i.e.* not a variable nor an abstraction), *e.g.* xx, $x(\lambda y.y)$, $x(yz)$ or xyz. As a consequence, there are *stuck* β*-redexes* such as $(\lambda y.t)(xx)$, *i.e.* β-redexes that will never be fired because their argument is normal, but it is not a value, nor will it ever become one. Such stuck β-redexes are a disease typical of (Naïve) Open CBV, but they spread to Strong CBV as well, because evaluating under abstraction forces to deal with locally open terms (*e.g.* the variable x is locally open with respect to $(\lambda y.t)(xx)$ in $s = \lambda x.((\lambda y.t)(xx))$, even if s is closed).

The real issue with stuck β-redexes is that they prevent the creation of other redexes, and provide *premature* β_v-normal forms. The issue is serious, as it can affect termination, and thus impact on notions of observational equivalence. Let $\delta = \lambda x.(xx)$. The problem is exemplified by the terms t and u in Eq. (1) below.

$$t := ((\lambda y.\delta)(zz))\delta \qquad u := \delta((\lambda y.\delta)(zz)) \tag{1}$$

In Naïve Open CBV, t and u are premature β_v-normal forms because they both have a stuck β-redex forbidding evaluation to keep going, while one would expect them to behave like the famous divergent term $\Omega := \delta\delta$ (see [2, 5, 12, 17, 25, 30] and pp. 4-5 in Sect. 2).

Open CBV Starting with the seminal work of Paolini and Ronchi Della Rocca [24, 25, 30], the dissonance between open terms and CBV has been repeatedly pointed out and studied *per se* via various calculi [2, 5, 7, 12, 15–17]. An important point is that the focus of most of these works is on Strong CBV. Studies about monad, CPS, and logical translations [14, 18, 22, 23, 31, 32] introduced further proposals.

These solutions inevitably extend β_v-reduction with some other rule(s) or constructor (as `let`-expressions) to deal with stuck β-redexes. They arise from different perspectives and each one has its pros and cons. By design, these calculi (when looked at in the context of Open CBV) are never observationally equivalent to Naïve Open CBV, as they all manage to (re)move stuck β-redexes and may diverge when Naïve Open CBV is instead stuck. Each one of these calculi, however, has its own notion of evaluation and normal form, and their mutual relationships are not evident.

The aim of this paper is to draw the attention of the community on Open CBV. We believe that it is somewhat deceiving that the

mainstream operational theory of CBV, however elegant, has to rely on closed terms, because it restricts the modularity of the framework, and raises the suspicion that the true essence of CBV has yet to be found. There is a real gap, indeed, between Closed and Strong CBV, as Strong CBV cannot be seen as an iteration of Closed CBV under abstractions because such an iteration has to deal with open terms. To improve the implementation of Coq [15], Grégoire and Leroy see Strong CBV as the iteration of the intermediate case of Open CBV, but they do not explore its theory. Here we exalt their point of view, providing a thorough operational study of Open CBV, as well as of the differences with respect to Closed/Strong CBV. Our motivations for insisting on Open CBV rather than Strong CBV are:

1. The issue with stuck β-redexes and premature β_v-normal forms is already visible in Open CBV;
2. Open CBV has a simpler rewriting theory than Strong CBV;
3. Our previous studies of Strong CBV in [5] and [12] naturally organized themselves as properties of Open CBV that were lifted to Strong CBV by a simple iteration under abstractions.

Our contributions are along two axes:

1. *Equivalence of the Proposals*: we show that the proposed generalizations of Naïve Open CBV are equivalent, in the sense that they have exactly the same sets of normalizing and diverging λ-terms. Therefore, *there is just one notion of Open CBV*, independently of its specific syntactic incarnation. We also compare the equational theories of the different proposals, and indicate the finest one for Open CBV.
2. *Cost Models and an Abstract Machine*: the termination results are complemented with quantitative analyses. Moreover we provide insights into the size-explosion problem for Closed/Open/Strong CBV, that lead to a new abstract machine for Open CBV, simpler than others in the literature.

Equivalence of the Proposals We focus on three proposals for Open CBV, as other solutions, *e.g.* Moggi's [23] or Herbelin and Zimmerman's [18], are already known to be equivalent to these ones (see the end of Sect. 2):

1. *The Fireball Calculus* $\lambda_{\texttt{fire}}$, that extends values to *fireballs* by adding so-called *inert terms* in order to restore harmony—it was introduced without a name by Paolini and Ronchi Della Rocca [25, 30], then rediscovered independently first by Leroy and Grégoire [15] to improve the implementation of Coq, and then by Accattoli and Sacerdoti Coen [7] to study cost models;
2. *The Value Substitution Calculus* $\lambda_{\texttt{vsub}}$, coming from the linear logic interpretation of CBV and using explicit substitutions and contextual rewriting rules to circumvent stuck β-redexes—it was introduced by Accattoli and Paolini [5] and it is a graph-free presentation of proof nets for the CBV λ-calculus [2];
3. *The Shuffling Calculus* $\lambda_{\texttt{sh}}$, that has rules to shuffle constructors, similar to Regnier's σ-rules for CBN [29], as an alternative to explicit substitutions—it was introduced by Carraro and Guerrieri [12] (and further analysed in [16, 17]) to study the adequacy of Open/Strong CBV with respect to denotational semantics related to linear logic.

The termination equivalences proved in the paper are more or less expected. Nonetheless we think they are interesting, at least for the following reasons:

1. *Uniform View*: we provide a new uniform view on a known problem, that will hopefully avoid further proliferations of CBV calculi for open/strong settings.
2. *Simple Rewriting Theory*: the relationships between the systems are developed using basic rewriting concepts. The technical development is simple, according to the best tradition of the CBV λ-calculus, and yet it provides a sharp and detailed decomposition of Open CBV evaluation.
3. *Connecting Different Worlds*: while $\lambda_{\texttt{fire}}$ is related to Coq and implementations, $\lambda_{\texttt{vsub}}$ and $\lambda_{\texttt{sh}}$ have a linear logic background. With respect to linear logic, $\lambda_{\texttt{vsub}}$ has been used for syntactical studies while $\lambda_{\texttt{sh}}$ for semantical ones. Our results therefore establish bridges between these different (sub)communities.

Equational Theories We compare the equational theories of these three calculi. In contrast to termination, the calculi are all equationally different. The theory of $\lambda_{\texttt{sh}}$ is strictly contained in that of $\lambda_{\texttt{vsub}}$—*i.e.* the one induced by linear logic proof nets—in turn strictly contained in the theory of $\lambda_{\texttt{fire}}$. We show, however, that the latter is not stable by context closure, *i.e.* it is not a congruence, and so it equates too much. Moreover, we show how to generalize $\lambda_{\texttt{sh}}$ so as to obtain exactly the same equational theory of $\lambda_{\texttt{vsub}}$, that is—in our opinion—*the* equational theory of Open CBV, also known to *strictly* contain Moggi's [23], see [5].

Cost Models and an Abstract Machine The number of β_v-steps is the canonical time cost model of Closed CBV, as first proved by Blelloch and Greiner [11, 13, 33]. In last year's LICS [7], Accattoli and Sacerdoti Coen generalized this result, showing that the number of steps in $\lambda_{\texttt{fire}}$ is a reasonable cost model for Open CBV. Here we show that the number of steps in $\lambda_{\texttt{vsub}}$ is linearly related to the number of steps in $\lambda_{\texttt{fire}}$, and so it provides a reasonable cost model as well. For $\lambda_{\texttt{sh}}$ we obtain a similar but strictly weaker result, due to some structural difficulties suggesting that the shuffling calculus is less apt to complexity analyses.

We then analyse the *size-explosion problem*, that is the degenerate behavior making the study of cost models for λ-calculi a delicate issue. We provide insights and refinements of some of the results of [7]. Our contributions with respect to cost models are:

1. *Open vs Strong Size-Explosion, a Simpler Abstract Machine*: we show that iterating Open CBV under abstractions introduces a malicious behavior that is not visible without iterations. In fact, the solution for such an issue is known from [7]. We show that the sophisticated techniques developed in [7] are in some sense not needed: if one tolerates a slight asymptotic slowdown (*i.e.* a quadratic rather than linear overhead with respect to the size of the initial term), a much simpler solution is possible. We then introduce an abstract machine, called Easy GLAMOUr, that is proved to be reasonable (*i.e.* its overhead is proved to be polynomial in the number of fireball steps) and that in contrast to the reasonable machines in [7] does not need to refine the environment with labels. We also provide a fine analysis of how different implementation techniques impact on the complexity of the corresponding machines.
2. *Minimality of the Cost Model*: the time cost model of Open CBV is the number of fireball steps. Fireballs extend values with inert terms, but reasonable implementations as the Easy GLAMOUr handle β-steps involving inert terms in constant time. It is natural to wonder if the cost model for Open CBV can simply be the number of steps *by value* (instead of *by fireball*, *i.e. by value* plus *by inert term*). We give a sort of negative result by exhibiting a family of terms that evaluate to normal form in n steps by value plus $O(2^n)$ steps by inert term, which shows that steps by inert term cannot be ignored, unless a radically different evaluation algorithm able to work up to inert redexes will be discovered.

Summing Up This paper is a collection of qualitative, quantitative, and implementative results about Open CBV. A further contribution, we believe, is the full picture, *i.e.* the recognition of Open CBV as a rich framework, meant to model a CBV discipline for the implementation of proof assistants, simpler than Strong CBV, and grounded in linear logic. From our comparative study it emerges that $\lambda_{\texttt{vsub}}$ is the most versatile of the studied calculi, and might be taken as the reference presentation of Open CBV. Different

Terms	$t, u, s, r ::= v \mid tu$
Values	$v, v' ::= x \mid \lambda x.t$
Evaluation Contexts	$E ::= \langle\cdot\rangle \mid tE \mid Et$

Rule at Top Level	Contextual closure
$(\lambda x.t)\lambda y.u \mapsto_{\beta_\lambda} t\{x \leftarrow \lambda y.u\}$	$E\langle t\rangle \to_{\beta_\lambda} E\langle u\rangle$ if $t \mapsto_{\beta_\lambda} u$
$(\lambda x.t)y \mapsto_{\beta_y} t\{x \leftarrow y\}$	$E\langle t\rangle \to_{\beta_y} E\langle u\rangle$ if $t \mapsto_{\beta_y} u$
Reduction	$\to_{\beta_v} := \to_{\beta_\lambda} \cup \to_{\beta_y}$

Figure 1. Naïve Open CBV $\lambda_{\texttt{Plot}}$

Terms and Values	As in Plotkin's Open CBV (Fig. 1)
Fireballs	$f, g, h ::= \lambda x.t \mid i$
Inert Terms	$i, i', i'' ::= x f_1 \dots f_n \quad n \geq 0$
Evaluation Contexts	$E ::= \langle\cdot\rangle \mid tE \mid Et$

Rule at Top Level	Contextual closure
$(\lambda x.t)(\lambda y.u) \mapsto_{\beta_\lambda} t\{x \leftarrow \lambda y.u\}$	$E\langle t\rangle \to_{\beta_\lambda} E\langle u\rangle$ if $t \mapsto_{\beta_\lambda} u$
$(\lambda x.t)i \mapsto_{\beta_i} t\{x \leftarrow i\}$	$E\langle t\rangle \to_{\beta_i} E\langle u\rangle$ if $t \mapsto_{\beta_i} u$
Reduction	$\to_{\beta_f} := \to_{\beta_\lambda} \cup \to_{\beta_i}$

Figure 2. The Fireball Calculus $\lambda_{\texttt{fire}}$

incarnations of Open CBV, however, serve different purposes, and the relationships established here provide a flexible framework to transfer concepts and result from one incarnation to the other.

Road map The next section provides an overview of the different presentations of Open CBV. Sect. 3 proves the termination equivalences, enriched with quantitative information, and Sect. 4 compares the equational theories. Sect. 5 discusses the size-explosion problem and how to circumvent it. Sect. 6 is devoted to the study of the Easy GLAMOUr. Last, Sect. 7 discusses the minimality of the cost model. Appendix A (p. 11) collects the definitions and notations of the rewriting notions at work in the paper.

Omitted proofs are in Appendix B (p. 11). In case of acceptance, this long version with Appendices will be made available on Arxiv.

2. Incarnations of Open Call-by-Value

Here we recall Naïve Open CBV $\lambda_{\texttt{Plot}}$ and introduce the three forms of Open CBV that will be compared ($\lambda_{\texttt{fire}}$, $\lambda_{\texttt{vsub}}$, and $\lambda_{\texttt{sh}}$) together with a semantic notion (*potential valuability*) reducing Open CBV to Closed CBV, and equivalent to normalization in $\lambda_{\texttt{fire}}$ and $\lambda_{\texttt{sh}}$ [12, 17, 30]. In this paper terms are always possibly open.

Naïve Open CBV: Plotkin's calculus $\lambda_{\texttt{Plot}}$ [28] Naïve Open CBV is Plotkin's weak CBV λ-calculus $\lambda_{\texttt{Plot}}$ on possibly open terms, defined in Fig. 1. Our presentation of the rewriting is unorthodox because we split β_v-reduction into two rules, according to the kind of value (abstraction or variable). The set of terms is denoted by Λ. Terms (in Λ) are always identified up to α-equivalence and the set of the free variables of a term t is denoted by $\texttt{fv}(t)$. We use $t\{x \leftarrow u\}$ for the term obtained by the capture-avoiding substitution of u for each free occurrence of x in t. Evaluation $\to_{\beta_v}$ is weak and non-deterministic, as in the case of an application there is no fixed order in the evaluation of the left and right subterms. As it is well-known, non-determinism is only apparent: the system is strongly confluent (see Appendix A for a glossary and notations of rewriting theory).

Proof p. 11 **Proposition 1.** *$\to_{\beta_y}$, $\to_{\beta_\lambda}$ and $\to_{\beta_v}$ are strongly confluent.*

Strong confluence is a remarkable property, much stronger than plain confluence. It implies that, given a term, all derivations to its normal form (if any) have the same length, and that normalization and strong normalization coincide, *i.e.* if there is a normalizing derivation then there are no diverging derivations. Strong confluence will also hold for $\lambda_{\texttt{fire}}$ and $\lambda_{\texttt{vsub}}$, not for $\lambda_{\texttt{sh}}$.

Let us come back to the splitting of $\to_{\beta_v}$. In Closed CBV it is well-known that $\to_{\beta_y}$ is superfluous, at least as long as small-step evaluation is considered, see [6] and Sect. 5. For Open CBV, $\to_{\beta_y}$ is instead necessary, but—as we explained in the introduction—it is not enough, which is why we shall consider extensions of $\lambda_{\texttt{Plot}}$. The main problem of Naïve Open CBV is that there are stuck β-redexes that break the harmony of the system. There are two kinds of solution, those *restoring a form of harmony*, to be thought as more semantical approaches, and those *removing stuck β-redexes*, that are more syntactical in nature.

Open Call-by-Value 1: The Fireball Calculus $\lambda_{\texttt{fire}}$ The Fireball Calculus $\lambda_{\texttt{fire}}$, defined in Fig. 2, was introduced without a name by Paolini and Ronchi Della Rocca in [25] and [30, Def. 3.1.4, p. 36] where its basic properties are also proved. We give here a presentation inspired by Accattoli and Sacerdoti Coen's [7], departing from it only for inessential, cosmetic details. Terms and evaluation contexts are the same as in $\lambda_{\texttt{Plot}}$.

The idea is to restore harmony by generalising $\to_{\beta_y}$ to fire when the argument is a more general *inert term*—the new rule is noted $\to_{\beta_i}$. The generalisation of values as to include inert terms is called *fireballs*. Actually fireballs and inert terms are defined by mutual recursion (in Fig. 2). For instance, $\lambda x.y$ is a fireball as an abstraction, while x, $y(\lambda x.x)$, xy, and $(z(\lambda x.x))(zz)(\lambda y.(zy))$ are fireballs as inert terms. Note that ii' is a inert term for every inert terms i and i'. Inert terms can be equivalently defined as $i ::= x \mid if$—such a definition is used in the proofs in the Appendix. Inert terms that are not variables are referred to as *compound inert terms*. The main feature of an inert term is that it is normal and that when plugged in a context it cannot create a redex, hence the name (it is not a so-called *neutral term* because it might have redexes under abstractions). In Grégoire and Leroy's presentation [15], inert terms are called *accumulators* and fireballs are simply called values.

Evaluation is given by the fireball rule $\to_{\beta_f}$, that is the union of $\to_{\beta_\lambda}$ and $\to_{\beta_i}$. For instance, consider $t := ((\lambda y.\delta)(zz))\delta$ and $u := \delta((\lambda y.\delta)(zz))$ as in Eq. (1), p. 1: t and u are β_v-normal but they diverge when evaluated in $\lambda_{\texttt{fire}}$, as desired: $t \to_{\beta_i} \delta\delta \to_{\beta_\lambda} \delta\delta \to_{\beta_\lambda} \dots$ and $u \to_{\beta_i} \delta\delta \to_{\beta_\lambda} \delta\delta \to_{\beta_\lambda} \dots$.

The distinguished, key property of $\lambda_{\texttt{fire}}$ is (for any $t \in \Lambda$)

Proposition 2 (Open Harmony). *t is β_f-normal iff t is a fireball.* Proof p. 11

The advantage of $\lambda_{\texttt{fire}}$ is its simple notion of normal form, *i.e.* fireballs, that have a clean syntactic description akin to that for call-by-name. Both $\lambda_{\texttt{vsub}}$ and $\lambda_{\texttt{sh}}$ will lack a nice, natural notion of normal form. The concepts of $\lambda_{\texttt{fire}}$, however, will allow us to somewhat identify a good notion of normal form also for $\lambda_{\texttt{vsub}}$.

The drawback of the fireball calculus—and probably the reason why its importance did not emerge before—is the fact that as a strong calculus it is not confluent: this is due to the fact that fireballs are not closed by substitution (see [30, p. 37]). Indeed, if evaluation is strong, the following critical pair cannot be joined, where $t := (\lambda y.I)(\delta\delta)$ and $I := \lambda z.z$ is the identity combinator:

$$I \,{}_{\beta_\lambda}\!\leftarrow (\lambda x.I)\delta \,{}_{\beta_i}\!\leftarrow (\lambda x.(\lambda y.I)(xx))\delta \to_{\beta_\lambda} t \to_{\beta_\lambda} t \to_{\beta_\lambda} \dots \quad (2)$$

Such a problem will play a role in Sect. 4, when we will discuss equational theories.

On the other hand, as long as evaluation is weak (that is the case we consider) everything works fine—the strong case can then be caught by iterating the weak one. In fact, fireball evaluation has a simple rewriting theory, as the next proposition shows. In particular it is strongly confluent.

Proposition 3 (Basic properties of $\lambda_{\texttt{fire}}$). Proof p. 12

1. *$\to_{\beta_i}$ is strongly normalizing and strongly confluent.*
2. *$\to_{\beta_\lambda}$ and $\to_{\beta_i}$ strongly commute.*

$$
\begin{array}{rrcl}
\texttt{vsub}\text{-Terms} & t,u,s & ::= & v \mid tu \mid t[x{\leftarrow}u] \\
\texttt{vsub}\text{-Values} & v & ::= & x \mid \lambda x.t \\
\text{Evaluation Contexts} & E & ::= & \langle\cdot\rangle \mid tE \mid Et \mid E[x{\leftarrow}u] \mid t[x{\leftarrow}E] \\
\text{Substitution Contexts} & L & ::= & \langle\cdot\rangle \mid L[x{\leftarrow}u]
\end{array}
$$

Rule at Top Level	Contextual closure
$L\langle\lambda x.t\rangle u \mapsto_{\mathsf{m}} L\langle t[x{\leftarrow}u]\rangle$	$E\langle t\rangle \to_{\mathsf{m}} E\langle u\rangle$ if $t \mapsto_{\mathsf{m}} u$
$t[x{\leftarrow}L\langle\lambda y.u\rangle] \mapsto_{\mathsf{e}_\lambda} L\langle t\{x{\leftarrow}\lambda y.u\}\rangle$	$E\langle t\rangle \to_{\mathsf{e}_\lambda} E\langle u\rangle$ if $t \mapsto_{\mathsf{e}_\lambda} u$
$t[x{\leftarrow}L\langle y\rangle] \mapsto_{\mathsf{e}_y} L\langle t\{x{\leftarrow}y\}\rangle$	$E\langle t\rangle \to_{\mathsf{e}_y} E\langle u\rangle$ if $t \mapsto_{\mathsf{e}_y} u$

Reductions $\quad \to_{\mathsf{e}} := \to_{\mathsf{e}_\lambda} \cup \to_{\mathsf{e}_y}, \quad \to_{\texttt{vsub}} := \to_{\mathsf{m}} \cup \to_{\mathsf{e}}$

Figure 3. The Value Substitution Calculus $\lambda_{\texttt{vsub}}$

3. *$\to_{\beta_f}$ is strongly confluent, and all β_f-normalizing derivations d from $t \in \Lambda$ (if any) have the same length $|d|_{\beta_f}$, the same number $|d|_{\beta_\lambda}$ of β_λ-steps, and the same number $|d|_{\beta_i}$ of β_i-steps.*

Rewriting Interlude: Creations of Type 1 and 4. The problem with stuck normal forms can be easily understood at the rewriting level as an issue about creations. According to Lévy [21], in the ordinary λ-calculus redexes can be created in 3 ways. Creations of type 1 take the following form

$$((\lambda x.\lambda y.t)r)s \to_\beta (\lambda y.t\{x{\leftarrow}r\})s$$

where the redex involving λy and s has been created by the β-step. Now, in Naïve Open CBV if r is a normal form that is not a value then the creation cannot take place, blocking evaluation. This is exactly the problem concerning the term t in Eq. (1), p. 1.

Actually, in CBV there also is a form of creation not considered by Lévy, let's call it of *type 4*:

$$(\lambda x.t)((\lambda y.v)v') \to_{\beta_v} (\lambda x.t)(v\{y{\leftarrow}v'\})$$

i.e. a reduction in the argument turns the argument itself into a value, creating a β_v-redex. As before, in a open setting v' may be replaced by a normal form that is not a value, blocking the creation of type 4. This is exactly the problem concerning the term u in Eq. (1), p. 1.

The following two proposals for Open CBV essentially introduce some way to enable creations of type 1 and 4, without substituting stuck β-redexes nor inert terms.

Open Call-by-Value 2: The Value Substitution Calculus $\lambda_{\texttt{vsub}}$ The *value substitution calculus* $\lambda_{\texttt{vsub}}$ of Accattoli and Paolini [2, 5] was introduced as a calculus for Strong CBV inspired by linear logic proof nets. In Fig. 3 we present its adaptation to Open CBV, obtained by simply removing abstractions from evaluation contexts. It extends the syntax of terms with the constructor $[x{\leftarrow}u]$, called *explicit substitution* (shortened ES, to not be confused with the meta-level substitution $\{x{\leftarrow}u\}$). A $\texttt{vsub}$-term $t[x{\leftarrow}u]$ represents the delayed substitution of u for x in t, *i.e.* stands for $\texttt{let}\ x = u\ \texttt{in}\ t$. So, $t[x{\leftarrow}u]$ binds the free occurrences of x in t. The set of $\texttt{vsub}$-terms—identified up to α-equivalence—is denoted by $\Lambda_{\texttt{vsub}}$ (clearly $\Lambda \subsetneq \Lambda_{\texttt{vsub}}$).

ES are used to remove stuck β-redexes: the idea is that β-redexes can be fired whenever—even if the argument is not a ($\texttt{vsub}$-)value—by means of the *multiplicative rule* $\to_{\mathsf{m}}$; however the argument is not substituted but placed in a ES. The actual substitution is done only when the content of the ES is a $\texttt{vsub}$-value, by means of the *exponential rule* $\to_{\mathsf{e}}$. These two rules are sometimes noted $\to_{\mathsf{dB}}$ (β at a distance) and $\to_{\mathsf{vs}}$ (substitution by value)—the names we use here are due to the interpretation of the calculus into linear logic proof-nets, see [2]. A characteristic feature coming from such an interpretation is that the rewriting rules are contextual, or *at a distance*: they are generalized as to act up to a list of substitutions (noted L, from *L*ist). Essentially, stuck β-redexes are turned into ES and then ignored by the rewriting rules—this is how creations of type 1 and 4 are enabled. For instance, the terms $t := ((\lambda y.\delta)(zz))\delta$ and $u := \delta((\lambda y.\delta)(zz))$ (as in Eq. (1), p. 1) are e-normal but $t \to_{\mathsf{m}} \delta[y{\leftarrow}zz]\delta \to_{\mathsf{m}} (xx)[x{\leftarrow}\delta][y{\leftarrow}zz] \to_{\mathsf{e}} (\delta\delta)[y{\leftarrow}zz] \to_{\mathsf{m}} (xx)[x{\leftarrow}\delta][y{\leftarrow}zz] \to_{\mathsf{e}} (\delta\delta)[y{\leftarrow}zz] \to_{\mathsf{m}} \dots$ and similarly for u.

The drawback of $\lambda_{\texttt{vsub}}$ is that it requires explicit substitutions.

The advantage of $\lambda_{\texttt{vsub}}$ is its simple and well-behaved rewriting theory, even simpler than the rewriting for $\lambda_{\texttt{fire}}$, as every rule terminates separately (while β_λ does not)—in particular strong confluence holds. Moreover, the theory has a sort of flexible second level given by a notion of structural equivalence, coming up next.

Proposition 4 (Basic Properties of $\lambda_{\texttt{vsub}}$, [5]). Proof p. 12

1. *$\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are strongly normalizing (separately).*
2. *$\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are strongly confluent (separately).*
3. *$\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ strongly commute.*
4. *$\to_{\texttt{vsub}}$ is strongly confluent, and all $\texttt{vsub}$-normalizing derivations d from $t \in \Lambda_{\texttt{vsub}}$ (if any) have the same length $|d|_{\texttt{vsub}}$, the same number $|d|_{\mathsf{e}}$ of e-steps, and the same number $|d|_{\mathsf{m}}$ of m-steps*
5. *Let $t \in \Lambda$. For any $\texttt{vsub}$-derivation d from t, $|d|_{\mathsf{e}} \leq |d|_{\mathsf{m}}$.*

Structural Equivalence The theory of $\lambda_{\texttt{vsub}}$ comes with a notion of structural equivalence $\equiv$, that equates $\texttt{vsub}$-terms that differ only for the position of ES. The basic idea is that the action of an ES via the exponential rule depends on the position of the ES itself only for inessential details (as long as the scope of binders is respected), namely the position of other ES, and thus can be abstracted away. A strong justification for the equivalence comes from the linear logic interpretation of the call-by-value λ-calculus, in which structurally equivalent $\texttt{vsub}$-terms translate to the same (recursively typed) proof net, see [2].

Structural equivalence $\equiv$ is defined as the least equivalence relation on $\Lambda_{\texttt{vsub}}$ closed by evaluation contexts (see Fig. 3) and generated by the following axioms:

$$
\begin{array}{rll}
t[y{\leftarrow}s][x{\leftarrow}u] \equiv_{\text{com}} t[x{\leftarrow}u][y{\leftarrow}s] & \text{if } y \notin \texttt{fv}(u) \text{ and } x \notin \texttt{fv}(s) \\
ts[x{\leftarrow}u] \equiv_{@\text{r}} (ts)[x{\leftarrow}u] & \text{if } x \notin \texttt{fv}(t) \\
t[x{\leftarrow}u]s \equiv_{@\text{l}} (ts)[x{\leftarrow}u] & \text{if } x \notin \texttt{fv}(s) \\
t[x{\leftarrow}u[y{\leftarrow}s]] \equiv_{[\cdot]} t[x{\leftarrow}u][y{\leftarrow}s] & \text{if } y \notin \texttt{fv}(t)
\end{array}
$$

We set $\to_{\texttt{vsub}_\equiv} := \equiv\!\to_{\texttt{vsub}}\!\equiv$ (*i.e.* for all $t,u \in \Lambda_{\texttt{vsub}}$: $t \to_{\texttt{vsub}_\equiv} r$ iff $t \equiv u \to_{\texttt{vsub}} s \equiv r$ for some $u,s \in \Lambda_{\texttt{vsub}}$). The notation $\to^+_{\texttt{vsub}_\equiv}$ keeps its usual meaning, while $\to^*_{\texttt{vsub}_\equiv}$ stands for $\equiv \cup \to^+_{\texttt{vsub}_\equiv}$, *i.e.* a $\texttt{vsub}_\equiv$-derivation of length zero can apply $\equiv$ and is not just the identity. As $\equiv$ is reflexive, $\to_{\texttt{vsub}} \subsetneq \to_{\texttt{vsub}_\equiv}$.

The rewriting theory of $\lambda_{\texttt{vsub}}$ enriched with structural equivalence $\equiv$ is remarkably simple, as the next lemma shows. In fact, $\equiv$ commutes with evaluation, and can thus be postponed. Additionally, the commutation is *strong*, as it preserves the number and kind of steps—one says that it is a *strong bisimulation* (with respect to $\to_{\texttt{vsub}}$). In particular, the equivalence is not needed to compute and it does not break, or make more complex, any property of $\lambda_{\texttt{vsub}}$. On the contrary, it enhances the equational theory and the flexibility of the system. It will be essential to establish simple and clean relationships with the other calculi for Open CBV.

Lemma 1 (Basic Properties of Structural Equivalence $\equiv$, [5]). *Let $t,u \in \Lambda_{\texttt{vsub}}$ and $\mathsf{x} \in \{\mathsf{m}, \mathsf{e}_\lambda\mathsf{e}_y, \mathsf{e}, \texttt{vsub}\}$.*

1. Strong Bisimulation of $\equiv$ wrt $\to_{\texttt{vsub}}$*: if $t \equiv u$ and $t \to_{\mathsf{x}} t'$ then there exists $u' \in \Lambda_{\texttt{vsub}}$ such that $u \to_{\mathsf{x}} u'$ and $t' \equiv u'$.*
2. Postponement of $\equiv$ wrt $\to_{\texttt{vsub}}$*: if $d\colon t \to^*_{\texttt{vsub}_\equiv} u$ then there are $s \equiv u$ and $e\colon t \to^*_{\texttt{vsub}} s$ such that $|d|_{\texttt{vsub}} = |e|_{\texttt{vsub}}$, $|d|_{\mathsf{e}} = |e|_{\mathsf{e}}$, $|d|_{\mathsf{e}_\lambda} = |e|_{\mathsf{e}_\lambda}$, $|d|_{\mathsf{e}_y} = |e|_{\mathsf{e}_y}$ and $|d|_{\mathsf{m}} = |e|_{\mathsf{m}}$.*
3. Normal Forms*: if $t \equiv u$ then t is x-normal iff u is x-normal.*
4. Strong confluence*: $\to_{\texttt{vsub}_\equiv}$ is strongly confluent.*

The first point is a variant of [5, Lemma 12], stating that $\equiv$ is a strong bisimulation, the other points are immediate consequences.

Terms and Values	As in Plotkin's Open CBV (Fig. 1)
Balanced Contexts	$B ::= \langle\cdot\rangle \mid tB \mid Bt \mid (\lambda x.B)t$

Rule at Top Level	Contextual Closure
$((\lambda x.t)u)s \mapsto_{\sigma_1} (\lambda x.ts)u,\ x \notin \mathtt{fv}(s)$	$B\langle t\rangle \to_{\sigma_1^\flat} B\langle u\rangle$ if $t \mapsto_{\sigma_1} u$
$v((\lambda x.s)u) \mapsto_{\sigma_3} (\lambda x.vs)u,\ x \notin \mathtt{fv}(v)$	$B\langle t\rangle \to_{\sigma_3^\flat} B\langle u\rangle$ if $t \mapsto_{\sigma_3} u$
$(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$	$B\langle t\rangle \to_{\beta_v^\flat} B\langle u\rangle$ if $t \mapsto_{\beta_v} u$

Reductions $\quad \to_{\sigma^\flat} := \to_{\sigma_1^\flat} \cup \to_{\sigma_3^\flat},\ \to_{\mathtt{sh}} := \to_{\beta_v^\flat} \cup \to_{\sigma^\flat}$

Figure 4. Shuffling λ-calculus $\lambda_{\mathtt{sh}}$

Open Call-by-Value 3: The Shuffling Calculus $\lambda_{\mathtt{sh}}$ The calculus introduced by Carraro and Guerrieri in [12], and here deemed *Shuffling Calculus*, has the same syntax of terms as Plotkin's calculus. Two additional commutation rules help $\to_{\beta_v}$ to deal with stuck β-redexes, by shuffling constructors so as to enable creations of type 1 and 4. As for $\lambda_{\mathtt{vsub}}$, $\lambda_{\mathtt{sh}}$ was actually introduced, and then used in [12, 16, 17], to study Strong CBV. In Fig. 4 we present its adaptation to Open CBV, based on *balanced contexts*, a special notion of evaluation contexts. The reductions $\to_{\sigma^\flat}$ and $\to_{\beta_v^\flat}$ are non-deterministic and—because of balanced contexts—can reduce under abstractions, but they are *morally* weak: they reduce under a λ only when the λ is applied to an argument. Note that the condition $x \notin \mathtt{fv}(s)$ (resp. $x \notin \mathtt{fv}(v)$) in the definition of the shuffling rule $\mapsto_{\sigma_1}$ (resp. $\mapsto_{\sigma_3}$) can always be fulfilled by α-conversion.

The reduction $\to_{\sigma^\flat}$ unblocks stuck β-redexes. For instance, consider the terms $t := ((\lambda y.\delta)(zz))\delta$ and $u := \delta((\lambda y.\delta)(zz))$ where $\delta := \lambda x.xx$ (as in Eq. (1), p. 1): t and u are $\beta_v^\flat$-normal but $t \to_{\sigma_1^\flat} (\lambda y.\delta\delta)(zz) \to_{\beta_v^\flat} (\lambda y.\delta\delta)(zz) \to_{\beta_v^\flat} \dots$ and $u \to_{\sigma_3^\flat} (\lambda y.\delta\delta)(zz) \to_{\beta_v^\flat} (\lambda x.\delta\delta)(zz) \to_{\beta_v^\flat} \dots$.

The similar shuffling rules in Open CBN, better known as Regnier's σ-rules [29], are *contained* in CBN β-equivalence, while in Open (and Strong) CBV they are more interesting, as they are not contained into (*i.e.* they enrich) β_v-equivalence.

The advantage of $\lambda_{\mathtt{sh}}$ is with respect to denotational investigations. In [12], $\lambda_{\mathtt{sh}}$ is indeed used to prove various semantical results in connection to linear logic, resource calculi, and the notion of Taylor expansion due to Ehrhard. In particular, in [12] it has been proved the adequacy of $\lambda_{\mathtt{sh}}$ with respect to the relational model induced by linear logic: a by-product of our paper is the extension of this adequacy result to all incarnations of Open CBV.

The drawback of $\lambda_{\mathtt{sh}}$ is its technical rewriting theory. We summarize some properties of $\lambda_{\mathtt{sh}}$, most of them proved in [12]:

Proof p. 14 **Proposition 5** (Basic Properties of $\lambda_{\mathtt{sh}}$, [12])**.**

1. *Let $t, u, s \in \Lambda$. If $t \to_{\beta_v^\flat} u$ and $t \to_{\sigma^\flat} s$ then $u \neq s$.*
2. *$\to_{\sigma^\flat}$ is strongly normalizing and (not strongly) confluent.*
3. *$\to_{\mathtt{sh}}$ is (not strongly) confluent.*
4. *Let $t \in \Lambda$: t is strongly $\mathtt{sh}$-normalizable iff t is $\mathtt{sh}$-normalizable.*

In contrast to $\lambda_{\mathtt{fire}}$ and $\lambda_{\mathtt{vsub}}$, $\lambda_{\mathtt{sh}}$ is not strongly confluent and not all $\mathtt{sh}$-normalizing derivations (if any) from a given term have the same length (consider, for instance, all $\mathtt{sh}$-normalizing derivations from $(\lambda y.z)(\delta(zz))\delta$). Nonetheless, normalization and strong normalization still coincide (Prop. 5.4), and Cor. 3 in Sect. 3 will show that the discrepancy is encapsulated inside the additional shuffling rules, as all $\mathtt{sh}$-normalizing derivations from a given term have the same number of $\beta_v^\flat$-steps.

Reducing Open to Closed Call-by-Value: Potential Valuability. Potential valuability relates Naïve Open CBV to Closed CBV via a meta-level substitution closing open terms: a (possibly open) term t is *potentially valuable* if there is a substitution of (closed) *values* for its free variables, for which it β_v-evaluates to a (closed) *value*.

In Naïve Open CBV, potentially valuable terms do not coincide with normalizable terms because of premature normal forms—as t and u in Eq. (1) at p. 1— which are not potentially valuable.

Paolini, Ronchi Della Rocca and, later, Pimentel [24–27, 30] gave several operational, logical, and semantical characterizations of potentially valuable terms in Naïve Open CBV. In particular, in [25, 30] Paolini and Ronchi Della Rocca prove that a term is potentially valuable in Plotkin's Naïve Open CBV iff its normalizable in $\lambda_{\mathtt{fire}}$.

Potentially valuable terms can be defined for every incarnation of Open CBV: it is enough to update the notions of evaluation and values in the above definition to the considered calculus. This has been done for $\lambda_{\mathtt{sh}}$ in [12], and for $\lambda_{\mathtt{vsub}}$ in [5]. For both calculi it has been proved that, in the weak setting, potentially valuable terms coincides with normalizable terms. In [17], it has been proved that Plotkin's potentially valuable terms coincide with $\lambda_{\mathtt{sh}}$-potentially valuable terms (which coincide in turn with $\mathtt{sh}$-normalizable terms). Our paper makes a further step: proving that termination coincides for $\lambda_{\mathtt{fire}}$, $\lambda_{\mathtt{vsub}}$, and $\lambda_{\mathtt{sh}}$ it implies that all their notions of potential valuability coincide with Plotkin's, *i.e.* there is just one notion of potential valuability.

There are other reasons why potential valuability is of interest. For instance, it is a key notion for characterizing solvability in Strong CBV [5, 12, 25, 30]. Moreover, in [26, 27] it has been proved that the class of strongly normalizable terms in Open CBN (aka the lazy λ-calculus, see [1]) coincides with that of potentially valuable terms (and that of normalizable terms in $\lambda_{\mathtt{fire}}$), providing an interesting connection between CBV and CBN.

Open CBV 4,5,6, ... Many calculi for Open CBV exist in the literature. Some of them have `let`-expressions (avatars of ES) and all of them have rules permuting constructors, therefore they lie somewhere in between $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$. Often, they have been developed for other purposes, usually to investigate the relationship with monad or CPS translations. Moggi's equational theory is a classic standard of reference, known to coincide with that of Sabry and Felleisen [31], Sabry and Wadler [32], Dychoff and Lengrand [14], Herbelin and Zimmerman [18] and Maraist et al's λ_{let} in [22]. In [5], $\lambda_{\mathtt{vsub}}$ modulo $\equiv$ is shown to be termination equivalent to Herbelin and Zimmerman's calculus, and to strictly contain its equational theory, and thus Moggi's. At the level of rewriting these presentations of Open CBV are all more involved than $\lambda_{\mathtt{vsub}}$, our reference presentation, because they do not rely on rules at a distance and so do not disentangle structural equivalence from evaluation. Their relationship with $\lambda_{\mathtt{vsub}}$ can be shown along the lines of that of $\lambda_{\mathtt{sh}}$ (or the one in [5]).

3. Quantitative Termination Equivalences

Here we show the equivalence with respect to termination of $\lambda_{\mathtt{fire}}$, $\lambda_{\mathtt{vsub}}$, and $\lambda_{\mathtt{sh}}$, enriched with quantitative information on the number of steps. The results are obtained simulating both $\lambda_{\mathtt{fire}}$ and $\lambda_{\mathtt{sh}}$ into $\lambda_{\mathtt{vsub}}$, which is the most flexible setting from a rewriting point of view. In both cases, structural equivalence $\equiv$ of $\lambda_{\mathtt{vsub}}$ plays a role.

Simulating $\lambda_{\mathtt{fire}}$ ***in*** $\lambda_{\mathtt{vsub}}$ A single β_v-step $(\lambda x.t)v \to_{\beta_v} t\{x \leftarrow v\}$ is simulated in $\lambda_{\mathtt{vsub}}$ by two steps (Lemma 2.1): $(\lambda x.t)v \to_{\mathsf{m}} t[x \leftarrow v] \to_{\mathsf{e}} t\{x \leftarrow v\}$, *i.e.* a m-step that creates a ES, and a e-step that turns the ES into the meta-level substitution performed by the β_v-step. The simulation of a inert step of $\lambda_{\mathtt{fire}}$ is instead trickier, because in $\lambda_{\mathtt{vsub}}$ there is no rule to substitute a inert term, if it is not a variable. The idea is that a inert step $(\lambda x.t)i \to_{\beta_i} t\{x \leftarrow i\}$ is simulated only by $(\lambda x.t)i \to_{\mathsf{m}} t[x \leftarrow i]$, *i.e.* only by the m-step that creates the ES, and such a ES will never be fired—so the simulation is up to the unfolding of substitutions containing inert terms (defined right next). Everything works because of the key property of inert terms: they are normal and their substitution cannot create redexes, so it is useless to substitute them.

The *unfolding* of a $\mathtt{vsub}$-term t is the term $t{\downarrow}$ obtained from t by turning ES into meta-level substitutions; it is defined by:

$$x{\downarrow} := x \qquad (tu){\downarrow} := t{\downarrow}\, u{\downarrow}$$
$$(\lambda x.t){\downarrow} := \lambda x.t{\downarrow} \qquad (t[x{\leftarrow}u]){\downarrow} := t{\downarrow}\{x{\leftarrow}u{\downarrow}\}$$

For all $t, u \in \Lambda_{\mathtt{vsub}}$, $t \equiv u$ implies $t{\downarrow} = u{\downarrow}$. Also, $t{\downarrow} = t$ iff $t \in \Lambda$.

In the simulation we are going to show, structural equivalence $\equiv$ plays a role. It is used to *clean* the $\mathtt{vsub}$-terms (with ES) obtained by simulation, putting them in a canonical form where ES do not appear among other constructors.

A $\mathtt{vsub}$-term is *clean* if it has the form $u[x_1{\leftarrow}i_1]\dots[x_n{\leftarrow}i_n]$ where $n \in \mathbb{N}$, $u \in \Lambda$ is called the *body*, and $i_1, \dots, i_n \in \Lambda$ are inert terms. Clearly, any term (as it is without ES) is clean.

We first show how to simulate a single fireball step.

Proof p. 15 **Lemma 2** (Simulation of a β_f-Step in $\lambda_{\mathtt{vsub}}$)**.** *Let $t, u \in \Lambda$.*

1. *If $t \to_{\beta_\lambda} u$ then $t \to_{\mathsf{m}} \to_{\mathsf{e}_\lambda} u$.*
2. *If $t \to_{\beta_i} u$ then $t \to_{\mathsf{m}} \equiv s$, with $s \in \Lambda_{\mathtt{vsub}}$ clean and $s{\downarrow} = u$.*

Unfortunately, it is not possible to simulate derivations by iterating Lemma 2, because the starting term t has no ES but the simulation of inert steps introduces ES. Therefore, we have to generalize the statement up to the unfolding of ES. In general, unfolding ES is a dangerous operation with respect to (non-)termination, as it may erase a diverging subterm (*e.g.* $t := x[y{\leftarrow}\delta\delta]$ is $\mathtt{vsub}$-divergent and $t{\downarrow} = x$ is normal). In our case, however, the simulation produces clean $\mathtt{vsub}$-terms, and so the unfolding is safe because it can only erase inert terms, that cannot create, erase, nor carry redexes.

By means of a technical lemma in the appendix we obtain:

Proof p. 15 **Lemma 3** (Projection of a β_f-Step on $\to_{\mathtt{vsub}}$ via Unfolding)**.** *Let t be a clean $\mathtt{vsub}$-term and u be a term.*

1. *If $t{\downarrow} \to_{\beta_\lambda} u$ then $t \to_{\mathsf{m}} \to_{\mathsf{e}_\lambda} s$, with $s \in \Lambda_{\mathtt{vsub}}$ clean and $s{\downarrow} = u$.*
2. *If $t{\downarrow} \to_{\beta_i} u$ then $t \to_{\mathsf{m}} \equiv s$, with $s \in \Lambda_{\mathtt{vsub}}$ clean and $s{\downarrow} = u$.*

Via Lemma 3 we can now simulate whole derivations. To obtain the termination equivalence, however, we have to work a little bit more. First of all, let us characterize the terms in $\lambda_{\mathtt{vsub}}$ obtained by projecting normalizing derivations (that always produce a fireball).

Proof p. 16 **Lemma 4.** *Let t be a clean $\mathtt{vsub}$-term. If $t{\downarrow}$ is a fireball, then t is $\{\mathsf{m}, \mathsf{e}_\lambda\}$-normal and its body is a fireball.*

Now, a $\{\mathsf{m}, \mathsf{e}_\lambda\}$-normal form t morally is $\mathtt{vsub}$-normal, as $\to_{\mathsf{e}_y}$ terminates (Prop. 4.1) and it cannot create $\{\mathsf{m}, \mathsf{e}_\lambda\}$-redexes. The part about creations is better expressed as a postponement property.

Proof p. 16 **Lemma 5** (Linear Postponement of $\to_{\mathsf{e}_y}$)**.** *Let t, u be $\mathtt{vsub}$-terms. If $d\colon t \to^*_{\mathtt{vsub}} u$ then $e\colon t \to^*_{\mathsf{m},\mathsf{e}_\lambda} \to^*_{\mathsf{e}_y} u$ with $|e|_{\mathtt{vsub}} = |d|_{\mathtt{vsub}}$, $|e|_{\mathsf{m}} = |d|_{\mathsf{m}}$, $|e|_{\mathsf{e}} = |d|_{\mathsf{e}}$ and $|e|_{\mathsf{e}_\lambda} \geq |d|_{\mathsf{e}_\lambda}$.*

The next theorem puts all the pieces together.

Proof p. 17 **Theorem 1** (Quantitative Simulation of $\lambda_{\mathtt{fire}}$ in $\lambda_{\mathtt{vsub}}$)**.** *Let $t, u \in \Lambda$. If $d\colon t \to^*_{\beta_f} u$ then there are $s, r \in \Lambda_{\mathtt{vsub}}$ and $e\colon t \to^*_{\mathtt{vsub}} r$ such that*

1. *Qualitative Relationship: $r \equiv s$, $u = s{\downarrow} = r{\downarrow}$ and s is clean;*
2. *Quantitative Relationship:*
   1. *Multiplicative Steps: $|d|_{\beta_f} = |e|_{\mathsf{m}}$;*
   2. *Exponential (Abstraction) Steps: $|d|_{\beta_\lambda} = |e|_{\mathsf{e}_\lambda} = |e|_{\mathsf{e}}$.*
3. *Normal Forms: if u is β_f-normal then there exists $f\colon r \to^*_{\mathsf{e}_y} q$ such that q is a $\mathtt{vsub}$-normal form and $|f|_{\mathsf{e}_y} \leq |e|_{\mathsf{m}} - |e|_{\mathsf{e}_\lambda}$.*

Corollary 1 (Linear Termination Equivalence of $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{fire}}$)**.**
Proof p. 17 *Let $t \in \Lambda$. There exists a β_f-normalizing derivation d from t iff there exists a $\mathtt{vsub}$-normalizing derivation e from t. Moreover, $|d|_{\beta_f} \leq |e|_{\mathtt{vsub}} \leq 2|d|_{\beta_f}$, i.e. they are linearly related.*

Note that the statement of Cor. 1 is stronger than it may look at first sight, because by strong confluence in both $\lambda_{\mathtt{fire}}$ and $\lambda_{\mathtt{vsub}}$, given a term t, if there is a normalizing derivation from t then there are no diverging derivations from t, and all normalizing derivations from t have the same length (Prop. 3.3 and Prop. 4.4).

Since the number of steps in $\lambda_{\mathtt{fire}}$ is known to be a reasonable cost model for Open CBV [7], our result states that also the number of steps in $\lambda_{\mathtt{vsub}}$ is a reasonable cost model, and moreover that they are tightly related. Not only the relationship between the two is linear, but the number of multiplicative steps in $\lambda_{\mathtt{vsub}}$ is *exactly* the number of steps in $\lambda_{\mathtt{fire}}$ (Thm. 1.2). By the way, this is somewhat surprising: in $\lambda_{\mathtt{fire}}$ arguments of β_f-redexes are required to be fireballs, while for m-redexes there are no restrictions on arguments, and yet in every derivation to normal form their number coincide.

By Lemma 4 it follows that a clean normal form is a fireball followed by ES with inert terms. This is a nice description of normal forms for $\lambda_{\mathtt{vsub}}$, inherited from $\lambda_{\mathtt{fire}}$, and a by-product of our study.

***Simulating* $\lambda_{\mathtt{sh}}$ *in* $\lambda_{\mathtt{vsub}}$** A derivation $d\colon t \to^*_{\mathtt{sh}} u$ in $\lambda_{\mathtt{sh}}$ is simulated via a projection on multiplicative normal forms in $\lambda_{\mathtt{vsub}}$ (for any $\mathtt{vsub}$-term t, its multiplicative normal form $\mathsf{m}(t)$ exists and is unique by Prop. 4), *i.e.* as a derivation $\mathsf{m}(t) \to^*_{\mathtt{vsub}_\equiv} \mathsf{m}(u)$. Indeed, a $\beta^\flat_v$-step of $\lambda_{\mathtt{sh}}$ is simulated in $\lambda_{\mathtt{vsub}}$ by a e-step followed by some m-steps to reach the m-normal form. Shuffling rules $\to_{\sigma^\flat}$ of $\lambda_{\mathtt{sh}}$ are simulated by the structural equivalence $\equiv$ of $\lambda_{\mathtt{vsub}}$: applying $\mathsf{m}(\cdot)$ to $((\lambda x.t)u)s \to_{\sigma^\flat_1} (\lambda x.(ts))u$ we obtain exactly an instance of the axiom $\equiv_{@l}$ defining $\equiv$: $\mathsf{m}(t)[x{\leftarrow}\mathsf{m}(u)]\mathsf{m}(s) \equiv_{@l} (\mathsf{m}(t)\mathsf{m}(s))[x{\leftarrow}\mathsf{m}(u)]$ (with the side conditions matching exactly). Similarly, $\to_{\sigma^\flat_3}$ projects to $\equiv_{@r}$ or $\equiv_{[\cdot]}$ (depending on whether v in $\to_{\sigma^\flat_3}$ is a variable or an abstraction). Therefore,

Lemma 6 (Projecting a $\mathtt{sh}$-Step on $\to_{\mathtt{vsub}_\equiv}$ via m-nf)**.** *Let $t, u \in \Lambda$.* Proof p. 18

1. *If $t \to_{\sigma^\flat} u$ then $\mathsf{m}(t) \equiv \mathsf{m}(u)$.*
2. *If $t \to_{\beta^\flat_v} u$ then $\mathsf{m}(t) \to_{\mathsf{e}} \to^*_{\mathsf{m}} \mathsf{m}(u)$.*

In contrast to the simulation of $\lambda_{\mathtt{fire}}$ in $\lambda_{\mathtt{vsub}}$, here the projection of a single step can be extended to derivations without problems, obtaining that the number of $\beta^\flat_v$-steps in $\lambda_{\mathtt{sh}}$ matches exactly the number of e-steps in $\lambda_{\mathtt{vsub}}$. Additionally, we apply the postponement of $\equiv$ (Lemma 1.2), factoring out the use of $\equiv$ (*i.e.* of shuffling rules) without affecting the number of e-steps. So, via Lemma 6 we can now simulate whole derivations. To obtain the termination equivalence, however, we need the following lemma:

Lemma 7 (Projection Preserves Normal Forms)**.** *Let $t \in \Lambda$. If t is* Proof p. 18 $\mathtt{sh}$*-normal then $\mathsf{m}(t)$ is $\mathtt{vsub}$-normal.*

The next theorem puts all the pieces together (for any $\mathtt{sh}$-derivation d, $|d|_{\beta^\flat_v}$ is the number of $\beta^\flat_v$-steps in d).

Theorem 2 (Quantitative Simulation of $\lambda_{\mathtt{sh}}$ in $\lambda_{\mathtt{vsub}}$)**.** *Let $t, u \in \Lambda$.* Proof p. 18 *If $d\colon t \to^*_{\mathtt{sh}} u$ then there are $s \in \Lambda_{\mathtt{vsub}}$ and $e\colon t \to^*_{\mathtt{vsub}} s$ such that*

1. *Qualitative Relationship: $s \equiv \mathsf{m}(u)$;*
2. *Quantitative Relationship (Exponential Steps): $|d|_{\beta^\flat_v} = |e|_{\mathsf{e}}$;*
3. *Normal Form: if u is $\mathtt{sh}$-normal then s and $\mathsf{m}(u)$ are $\mathtt{vsub}$-normal.*

Corollary 2 (Termination Equivalence of $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$)**.** *Let $t \in \Lambda$.* Proof p. 18 *There is a $\mathtt{sh}$-normalizing derivation d from t iff there is a $\mathtt{vsub}$-normalizing derivation e from t. Moreover, $|d|_{\beta^\flat_v} = |e|_{\mathsf{e}}$.*

As for Cor. 1, the claim of Cor. 2 is stronger than it seems, since for both $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$, given a term t, if there is a normalizing derivation from t then there are no diverging derivations from t (for $\lambda_{\mathtt{vsub}}$ it follows from strong confluence, for $\lambda_{\mathtt{sh}}$ is given by Prop. 5.4).

About the quantitative relationship, $|d|_{\beta^\flat_v} = |e|_{\mathsf{e}}$ also holds for all normalizing derivations from a given term; for $\lambda_{\mathtt{vsub}}$, it holds by Prop. 4.4; for $\lambda_{\mathtt{sh}}$, it is given by the following corollary of Thm. 2.

Corollary 3 (Number of $\beta^\flat_v$-Steps is Invariant)**.** *All $\mathtt{sh}$-normalizing* Proof p. 18

derivations from $t \in \Lambda$ (if any) have the same number of $\beta_v^\flat$-steps.

In a way, the quantitative simulation of $\lambda_{\mathtt{sh}}$ in $\lambda_{\mathtt{vsub}}$ (Thm. 2) "imposes the good behavior" of $\lambda_{\mathtt{vsub}}$ on $\lambda_{\mathtt{sh}}$. The existence of a quantitative invariant in sh-normalizing derivations is not obvious, indeed, as $\lambda_{\mathtt{sh}}$ is not strongly confluent.

For what concerns the cost model things are subtler for $\lambda_{\mathtt{sh}}$. Note that the relationship between $\lambda_{\mathtt{sh}}$ and $\lambda_{\mathtt{vsub}}$ uses the number of e-steps, while the cost model (inherited from $\lambda_{\mathtt{fire}}$) is the number of m-steps. Do e-steps provide a reasonable cost model? Sect. 7 addresses this question, showing that it is quite unlikely. The next section will also say a little bit more about the cost model for $\lambda_{\mathtt{sh}}$.

4. Equational Theories

Here we compare the equational theories of $\lambda_{\mathtt{fire}}$, $\lambda_{\mathtt{vsub}}$ (with and without $\equiv$), and $\lambda_{\mathtt{sh}}$, *i.e.* the reflexive-transitive and symmetric closures $\simeq_{\mathsf{r}}$ of $\to_{\mathsf{r}}$ for $r \in \{\beta_f, \mathtt{vsub}, \mathtt{vsub}_\equiv, \mathtt{sh}\}$.

The Theory of $\lambda_{\mathtt{fire}}$ is Too Large It is easy to adapt the counterexample to confluence for Strong CBV in $\lambda_{\mathtt{fire}}$ (Sect. 2) to an example that $\simeq_{\beta_f}$ is larger than the contextual equivalence of $\lambda_{\mathtt{fire}}$, thus showing that it is an inadequate equational theory. Consider $t := (\lambda y.I)(xx)$ and $u := (\lambda y.I)(xz)$ where I is the identity combinator. We have $t \simeq_{\beta_f} u$, as they both β_i-reduce to I. Despite their equivalence, the context $C := (\lambda z.((\lambda x.\langle\cdot\rangle)\delta))I$, where δ is the duplicator combinator, separates them, and so $C\langle t\rangle \not\simeq_{\beta_f} C\langle u\rangle$. In fact, $C\langle t\rangle \to^2_{\beta_\lambda} (\lambda y.I)(\delta\delta)$, that diverges, while $C\langle u\rangle \to^*_{\beta_\lambda} I$. The problem is that $\lambda_{\mathtt{fire}}$ erases too much. Note that on the other hand $t \not\simeq_{\mathtt{vsub}_\equiv} u$, as their normal forms in $\lambda_{\mathtt{vsub}}$ are $I[y{\leftarrow}xx] \not\equiv I[y{\leftarrow}xz]$. From Thm. 1 it also easily follow the following result.

Proof p. 19 **Proposition 6.** *$\simeq_{\mathtt{vsub}_\equiv}$ is contained in $\simeq_{\beta_f}$ on normalizable terms.*

We did not dig on diverging terms because the question is quite more technical and—more generally—the example we showed suggests that $\simeq_{\beta_f}$ is not worth to be studied in detail.

The Theory of $\lambda_{\mathtt{sh}}$ is Strictly Contained in the Theory of $\lambda_{\mathtt{vsub}}$ Up to $\equiv$ From the simulation of $\lambda_{\mathtt{sh}}$ into $\lambda_{\mathtt{vsub}}$ (Thm. 2), it follows that $\simeq_{\mathtt{sh}} \subseteq \simeq_{\mathtt{vsub}_\equiv}$, but the converse does not hold on Λ. It is easy to understand why: the projection used in the simulation sends $\sigma_1^\flat$ to $\equiv_{@\mathsf{l}}$ and $\sigma_3^\flat$ to $\equiv_{@\mathsf{r}} \,/\, \equiv_{[\cdot]}$, but $\equiv_{\mathsf{com}}$ is not covered. By turning ES into β-redexes we can easily turn $\equiv_{\mathsf{com}}$ into a pair of terms that are equivalent for $\simeq_{\mathtt{vsub}_\equiv}$ but not for $\simeq_{\mathtt{sh}}$. Let i and i' two distinct compound inert terms and consider $s := (\lambda y.((\lambda x.z)i))i' \simeq_{\mathtt{vsub}_\equiv} (\lambda x.((\lambda y.z)i'))i =: r$ that are such that $s \not\simeq_{\mathtt{sh}} r$, since s and r are sh-normal and different. Actually, there is a further slight mismatch. Note that the definition of $\to_{\sigma_3^\flat}$ requires the presence of a value v, which is absent from $\equiv_{@\mathsf{r}}$ and $\equiv_{[\cdot]}$. Such a restriction on $\to_{\sigma_3^\flat}$ is required to have confluence for $\to_{\sigma^\flat}$ (the unrestricted $\sigma^\flat$-rules are not confluent on *e.g.* $(Ii)(Ii')$). By lifting a general case of $\equiv_{@\mathsf{r}}$ as we did for $\equiv_{\mathsf{com}}$, one obtains a pair of terms that are equivalent for $\simeq_{\mathtt{vsub}_\equiv}$, but not for $\simeq_{\mathtt{sh}}$.

Extending $\simeq_{\mathtt{sh}}$ to Match $\simeq_{\mathtt{vsub}_\equiv}$ According to the previous paragraph, to close the gap between $\simeq_{\mathtt{sh}}$ and $\simeq_{\mathtt{vsub}_\equiv}$ is enough to extend $\simeq_{\mathtt{sh}}$ with an equation corresponding to the lifting of $\equiv_{\mathsf{com}}$ and by taking the more general form of $\to_{\sigma_3^\flat}$. Consider:

$$\begin{array}{ll} (\lambda y.(\lambda x.t)u)s \mapsto_{\sigma_{\mathsf{com}}} (\lambda x.(\lambda y.t)s)u & \text{if } y \notin \mathtt{fv}(u),\ x \notin \mathtt{fv}(s) \\ t((\lambda x.s)u) \mapsto_{\sigma'_3} (\lambda x.ts)u & \text{if } x \notin \mathtt{fv}(t). \end{array}$$

and define the *extended shuffling equivalence* $\simeq^{\mathsf{ext}}_{\mathtt{sh}}$ as the least equivalence relation on Λ closed under balanced context (see Fig. 4) containing $\mapsto_{\sigma_{\mathsf{com}}}$, $\mapsto_{\sigma_1}$, $\mapsto_{\sigma'_3}$ and $\mapsto_{\beta_v}$. Clearly, $\simeq_{\mathtt{sh}} \subsetneq \simeq^{\mathsf{ext}}_{\mathtt{sh}}$.

It easy to show that $\simeq^{\mathsf{ext}}_{\mathtt{sh}}$ is contained in $\simeq_{\mathtt{vsub}_\equiv}$ by projecting on m-normal forms. The other direction is shown by defining a translation $(\cdot)^\circ$ of vsub-terms into terms (without ES) that is a sort of inverse of the multiplicative projection, as it turns ES into β-redexes:

$$\begin{array}{ll} x^\circ := x & (tu)^\circ := t^\circ u^\circ \\ (\lambda x.t)^\circ := \lambda x.t^\circ & (t[x{\leftarrow}u])^\circ := (\lambda x.t^\circ)u^\circ \end{array}$$

For any $t \in \Lambda_{\mathtt{vsub}}$, the number of ES occurring in t is denoted by $|t|_{\mathsf{ES}}$. Clearly, $t^\circ = t$ iff $t \in \Lambda$ iff $|t|_{\mathsf{ES}} = 0$ (recall that $\Lambda \subsetneq \Lambda_{\mathtt{vsub}}$).

Lemma 8 (Projection of $\mathtt{vsub}_\equiv$ on $\lambda_{\mathtt{sh}}$)**.** *Let $t, u \in \Lambda_{\mathtt{vsub}}$.* Proof p. 19

1. *If $t \to_{\mathsf{m}} u$ then $t^\circ \to^n_{\sigma_1^\flat} u^\circ$ with $n \le |t|_{\mathsf{ES}}$.*
2. *If $t \to_{\mathsf{e}} u$ then $t^\circ \to^n_{\sigma_3^\flat} \to_{\beta_v^\flat} u^\circ$ where $n \le |t|_{\mathsf{ES}}$.*
3. *If $t \equiv u$ then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$.*
4. *If $t \to_{\mathtt{vsub}_\equiv} u$ then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$.*

Theorem 3 (Same Equational Theory for $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$)**.** *Let $t, u \in \Lambda$: $t \simeq_{\mathtt{vsub}_\equiv} u$ iff $t \simeq^{\mathsf{ext}}_{\mathtt{sh}} u$.* Proof p. 20

We can also say something on the theory of $\lambda_{\mathtt{vsub}}$ without $\equiv$, namely that it is strictly contained in the theory of $\lambda_{\mathtt{sh}}$. The containment is given by Lemmas 8.1-2, and strictness by the fact that $\sigma^\flat$-rules are not simulable on $\lambda_{\mathtt{vsub}}$ without $\equiv$. For instance, $t := (\lambda x.y)(xx)z \simeq_{\mathtt{sh}} (\lambda x.yz)(xx) =: u$, but t and u have two different normal forms in vsub.

The Theory of Open CBV We choose $\simeq_{\mathtt{vsub}_\equiv}$ as a reference theory for Open CBV. It is a simple, modular *two-levels* theory, where evaluation and administrative differences, encapsulated into structural equivalence, are disentangled. It strictly contains Moggi's theory (and thus subsumes also the ones in [14, 18, 22, 31, 32]), it coincides with the theory induced by linear logic proof nets [2], it avoids the degeneracies of $\lambda_{\mathtt{fire}}$, can be projected on $\lambda_{\mathtt{sh}}$, and it has a nice notion of clean normal form.

Two More Words About Cost Models for $\lambda_{\mathtt{sh}}$ Note the bound $n \le |t|_{\mathsf{ES}}$ in Lemmas 8.1-2. It is easily seen that given a λ-term u and a derivation $d\colon u \to^*_{\mathtt{vsub}} t$ then $|t|_{\mathsf{ES}}$ is equal to the number $|d|_{\mathsf{m}}$ of m-steps in d. One can then prove that for any derivation $d\colon u \to^*_{\mathtt{vsub}} t$ there is a derivation $e\colon u^\circ \to^*_{\mathtt{sh}} t^\circ$ such that $|e|_{\sigma^\flat} = O(|d|^2_{\mathsf{m}})$ thus establishing a polynomial relationship with the cost model of $\lambda_{\mathtt{vsub}}$. Since $\lambda_{\mathtt{sh}}$ is not strongly confluent, however, this fact is quite weak without further analysis, as it does not extend to all derivations in $\lambda_{\mathtt{sh}}$. The moral is that $\lambda_{\mathtt{sh}}$ is not really apt to study the complexity of evaluation of Open CBV.

5. How to Stop Worrying and Love the Bomb

It is well-known that Closed CBV admits *size-exploding families*, *i.e.* families where the n-th term has size linear in n and in n β_v-steps evaluates to a term of size exponential in n. The following is an interesting example. Fix a closed value v (*e.g.* the identity). Define

$$\begin{array}{ll} t_1 := \lambda x.\lambda y.(yxx) & R_0^v := v \\ t_{n+1} := \lambda x.(t_n(\lambda y.(yxx))) & R_{n+1}^v := \lambda y.(yR_n^vR_n^v) \end{array}$$

Proposition 7 (Abstraction Size-Explosion)**.** *For all $n > 0$, $t_n v \to^n_{\beta_\lambda} R_n^v$ with $|t_n| = O(n)$, $|R_n^v| = O(2^n)$, and R_n^v is β_f-normal.* Proof p. 20

This family is interesting because no matter how one looks at it, it always explodes: in Closed/Open CBV there is only one possible derivation to normal form and in Strong CBV/CBN all such derivations have the same length (and are permutatively equivalent). To our knowledge this family never appeared in print.

The issue is that the number of β_v-steps does not seem to be a reasonable cost model, as it does not even account for the time of writing down the normal form. It is also well-known, however, that adding sharing and turning to *micro-step evaluation* (under the form of abstract machines, graph-rewriting, or explicit substitutions) allows to circumvent such a problem in Closed CBV, as in a

number of micro steps linear in the number of β-steps one reaches a reasonable compact normal form, *i.e.* a shared representation of the normal form that has size linear in n and that can be managed efficiently. This is possible because with micro-step evaluation variable occurrences under abstraction are never substituted by values: note that in R_n^v there is an exponential number of copies of v *under abstraction*.

Turning to Open CBV another form of size-explosion appears. Fix a inert term i. Define:

$$u_1 := \lambda x_1.(x_1 x_1) \qquad u_{n+1} := \lambda x_{n+1}.(u_n(x_{n+1}x_{n+1}))$$

Proof p. 20 **Proposition 8** (Inert Size-Explosion [7]). *For all $n>0$, $u_n i \to_{\beta_i}^n i^{2^n}$ (the β_f-normal form of $u_n i$) with $|u_n| = O(n)$ and $|i^{2^n}| = O(2^n)$.*

As we show in Sect. 6, in order to circumvent inert size-explosion it is enough to *never substitute inert terms* at the micro-step level.

In Strong CBV one has both abstraction and inert size-explosion, but there is worse. The way the abstraction case is circumvented in micro-step Closed CBV does not work for Strong CBV, because the exponential number of values v appearing under abstraction in the result R_n^v (and causing the explosion) has to be substituted in order to obtain a strong normal form. The solution developed in [7] (and inspired by [3]) is to *substitute abstractions on-demand* (in addition to *never substitute inert terms*): the substitution of an abstraction happens only if the variable occurrence that it should replaces is applied, so that a β-redex is created.

Now, in [3] the solution for Strong CBV is developed in the context of Open CBV. The contribution we give here is to show that *substituting abstractions on-demand* is not necessary for Open CBV: freely substituting abstractions still provides a reasonable (and simpler) implementation, if one only cares about Open CBV. Said differently, iterating Open CBV to catch Strong CBV has non-trivial implications at the level of complexity, and schemes that are reasonable for Open CBV might induce—when iterated—non reasonable schemes for Strong CBV. But there is quite more.

An implementation of Open CBV is *reasonable* when given a derivation $d : t \to_{\beta_f}^k u$ the complexity of its implementation is polynomial with respect to two parameters, the number k of steps (*i.e.* the cost model) and the size $|t|$ of the initial term (roughly the input). In [3] it is shown that *substituting abstractions on-demand* and *never substituting compound inert terms* provides an implementation of complexity $O(k^2 \cdot |t|)$, and that if in addition one never substitutes variables (thus simply *never substituting inert terms*) the complexity lowers to $O(k \cdot |t|)$, becoming bilinear. The machine of the next section will only *never substitute inert terms* and will have complexity $O(k \cdot |t|^2)$, shifting the quadratic dependency on the size of the initial term. We omit it for space reasons, but it is easy to design a machine that *never substitutes compound inert terms* and that has complexity $O(k^2 \cdot |t|^2)$. The moral is that

For implementations of Open CBV *never substituting variables* and *substituting abstractions on-demand* are modular optimizations that act on separate parameters of the overhead, reducing the bound from quadratic to linear.

6. Easy GLAMOUr

In this section we present the Easy GLAMOUr, a simplified version of the GLAMOUr machine from [7], not needing any labeling of codes and yet providing a reasonable implementation. For a comparison with Grégoire and Leroy [15] see Sect. II of [7].

Background GLAMOUr stays for *Useful* (*i.e.* optimized to be reasonable) *Open* (reducing open terms) *Global* (using a single global environment) LAM, and LAM stays for *Leroy Abstract Machine*, an ordinary machine implementing left-to-right call-by-value, defined in [8]. In [7] the study of the GLAMOUr was done according to the distillation approach of [8], *i.e.* by decoding the machine towards a micro-step λ-calculus with ES. Here we follow the distillation approach only partially: we borrow the terminology, but we decode directly to $\lambda_{\mathtt{fire}}$, which is simpler.

Right-to-Left Evaluation The operational semantics of $\lambda_{\mathtt{fire}}$ defined in Sect. 2 is non-deterministic. We fix a deterministic strategy, the *right-to-left evaluation* $\to_{\mathtt{r}\beta_f}$, defined by closing the root rules $\mapsto_{\beta_\lambda}$ and $\mapsto_{\beta_i}$ in Fig. 2 by *right contexts*, given by $R ::= \langle\cdot\rangle \mid tR \mid Rf$. The next lemma guarantees our definition is correct.

Lemma 9 (Properties of $\to_{\mathtt{r}\beta_f}$). *Let $t \in \Lambda$.* Proof p. 20

1. Completeness*: t has $\to_{\beta_f}$-redex iff t has a $\to_{\mathtt{r}\beta_f}$-redex.*
2. Determinism*: t has at most one $\to_{\mathtt{r}\beta_f}$-redex.*

Machine Components The Easy GLAMOUr, defined in Table 1, implements $\to_{\mathtt{r}\beta_f}$ via a decoding function $\underline{\cdot}$ mapping machine states to λ-terms. A machine state s is a quadruple given by

- *Code* $\bar{t}$: it is a term without ES not considered up to α-equivalence, which is why it is over-lined;
- *Stack* π: it contains the arguments of the current code. Note that stacks items ϕ are pairs $x@\pi$ and $\lambda x.\overline{u}@\epsilon$. These pairs allow to implement some of the transitions in constant time. The pair $x@\pi$ codes the term $\underline{\pi}\langle x\rangle$ that would be obtained by putting x in the context obtained by decoding the stack π. The pair $\lambda x.\overline{u}@\epsilon$ is used to inject abstractions into pairs, so that items ϕ can be uniformly seen as pairs $\bar{t}@\pi$ of a code $\bar{t}$ and a stack π;
- *Dump* D: a second stack, that together with the stack π is used to walk through the term and search for the next redex to reduce. The dump is extended with an entry $\bar{t}\lozenge\pi$ every time evaluation enters in the right subterm of an application. The entry saves the left part of the application (the code $\bar{t}$) and the current stack, to restore them when the evaluation of the right subterm is over.
- *Global Environment* E: it is used to implement micro-step evaluation (*i.e.* the substitution on a variable occurrence at the time), storing the ES that have been created so far (when fireball redexes were encountered). Most of the literature on abstract machines uses *local environments* and *closures*. Having just one global environment removes the need for closures and simplifies the machine. On the other hand, it forces to use explicit α-renamings (in $\rightsquigarrow_{\mathtt{e}}$), but this does not affect the overall complexity, as it speeds up other operations, see [8]. We write $E(x) = \bot$ when in E there are no entries of the form $[x{\leftarrow}\phi]$. To save space, sometimes we write $[x{\leftarrow}\bar{t}]E$ for $[x{\leftarrow}\bar{t}] : E$.

The Decoding Every state s decodes to a term $\underline{s}$ (see the top right part of Table 1), having the shape $\underline{E_s}\langle\bar{t}\sigma_E\rangle$, where $\bar{t}\sigma_E$ is a λ-term, obtained by applying to the code the meta-level substitution σ_E induced by the global environment E, and $\underline{E_s}$ is an evaluation context, obtained by decoding the stack π and the dump D and then applying σ_E. Note that, to improve readability, stacks are decoded to contexts in postfix notation for plugging.

The Transitions The union of the transitions of the Easy GLAMOUr is noted $\rightsquigarrow$. According to the distillation approach we distinguish different kinds of transitions, whose names reflect a proof-theoretical view, as machine transitions can be seen as cut-elimination steps [8, 9]:

- *Multiplicative* $\rightsquigarrow_{\mathtt{m}}$: it morally fires a $\to_{\mathtt{r}\beta_f}$-redex, except that its action puts a new ES in the environment instead of substituting the argument, as $\to_{\mathtt{m}}$ in $\lambda_{\mathtt{vsub}}$;
- *Exponential* $\rightsquigarrow_{\mathtt{e}}$: performs a clashing-avoiding substitution from the environment on the single occurrence represented by the current code. It is a micro-step variant of rule $\to_{\mathtt{e}_\lambda}$ of $\lambda_{\mathtt{vsub}}$.
- *Commutative transitions*, all together noted $\rightsquigarrow_{\mathtt{c}}$: they locate and expose the next redex according to the right-to-left strategy,

Table 1. Easy GLAMOUr machine: data-structures (stacks π, dumps D, global env. E, states s), decoding $\underline{\cdot}$, and transitions

$$\begin{array}{llll} \phi & ::= & \lambda x.\overline{u}@\epsilon \mid x@\pi & E ::= \epsilon \mid [x\leftarrow\phi] : E \\ \pi & ::= & \epsilon \mid \phi : \pi & s ::= (D, \bar{t}, \pi, E) \\ D & ::= & \epsilon \mid D : \bar{t}\diamondsuit\pi & \end{array}$$

$$\begin{array}{lcll} \underline{\epsilon} & := & \langle\cdot\rangle & \sigma_{[x\leftarrow\phi]:E} := \{x\leftarrow\underline{\phi}\}\sigma_E \\ \underline{\phi : \pi} & := & \langle\langle\cdot\rangle\underline{\phi}\rangle\underline{\pi} & \underline{E_s} := \underline{D\langle\overline{\pi}\rangle}\sigma_E \\ \underline{t@\pi} & := & \langle t\rangle\underline{\pi} & \underline{s} := \underline{E_s\langle\bar{t}\sigma_E\rangle} \\ \underline{D : \bar{t}\diamondsuit\pi} & := & \underline{D}\langle\langle\bar{t}\langle\cdot\rangle\rangle\underline{\pi}\rangle & \text{where } s = (D, \bar{t}, \overline{\pi}, E) \end{array}$$

Dump	Code	Stack	Global Env		Dump	Code	Stack	Global Env	
D	$\bar{t}\overline{u}$	π	E	$\rightsquigarrow_{c_1}$	$D : \bar{t}\diamondsuit\pi$	$\overline{u}$	ϵ	E	
$D : \bar{t}\diamondsuit\pi$	$\lambda x.\overline{u}$	ϵ	E	$\rightsquigarrow_{c_2}$	D	$\bar{t}$	$\lambda x.\overline{u}@\epsilon : \pi$	E	
$D : \bar{t}\diamondsuit\pi$	x	π'	$E_1[x\leftarrow y@\pi'']E_2$	$\rightsquigarrow_{c_3}$	D	$\bar{t}$	$x@\pi' : \pi$	$E_1[x\leftarrow y@\pi'']E_2$	
$D : \bar{t}\diamondsuit\pi$	x	π'	E	$\rightsquigarrow_{c_4}$	D	$\bar{t}$	$x@\pi' : \pi$	E	if $E(x) = \bot$
D	$\lambda x.\bar{t}$	$\phi : \pi$	E	$\rightsquigarrow_{\mathtt{m}}$	D	$\bar{t}$	π	$[x\leftarrow\phi]E$	
D	x	π	$E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2$	$\rightsquigarrow_{\mathtt{e}}$	D	$(\lambda y.\overline{u})^\alpha$	π	$E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2$	

where $(\lambda y.\overline{u})^\alpha$ is any code α-equivalent to $\lambda y.\overline{u}$ such that its bound names are distinct and fresh with respect to those in D, π and $E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2$.

by rearranging the data-structures. They are invisible on the calculus. The commutative rule $\rightsquigarrow_{c_1}$ forces evaluation to be right-to-left on applications: the machine processes first the right subterm $\overline{u}$, saving the left sub term $\bar{t}$ on the dump together with its current stack π. The role of $\rightsquigarrow_{c_2}$, $\rightsquigarrow_{c_3}$, and $\rightsquigarrow_{c_4}$ is to backtrack to the entry on top of the dump. When the right subterm, *i.e.* the pair $\bar{t}@\pi$ of current code and stack, is finally in normal form, it is pushed on the stack and the machine backtracks. Note $\rightsquigarrow_{c_3}$: inert terms are never substituted.

Garbage collection is here simply ignored, or, more precisely, it is encapsulated at the meta-level, in the decoding function.

The Weak Bisimulation The machine starts executions on *initial states* of the form $(\epsilon, \bar{t}, \epsilon, \epsilon)$, where $\bar{t}$ is such that any two variables (bound or free) have distinct names, and any other component is empty. A state s is *reachable* if there are an initial state s' and an execution $\rho : s' \rightsquigarrow^* s$, and it is *final* if no transitions apply.

The study of the machine relies on the following invariants.

Proof p. 20 **Lemma 10** (Easy GLAMOUr Invariants)**.** *Let $s = (D, \bar{t}, \pi, E)$ be a reachable state. Then:*

1. Name:
   1. Substitutions*: if $E = E' : [x\leftarrow\overline{u}] : E''$ then x is fresh wrt $\overline{u}$ and E'';*
   2. Abstractions*: if $\lambda x.\bar{s}$ is a subterm of D, $\overline{u}$, π, or E then x may occur only in $\bar{s}$;*
2. Fireball Item*: $\underline{\phi\sigma_E}$ is a inert term if $\phi = x@\pi'$ and an abstraction otherwise, for every item ϕ in π, in E, and in every stack in D;*
3. Contextual Decoding*: $E_s = \underline{D\langle\overline{\pi}\rangle\sigma_E}$ is a right context;*

The invariants are used to prove the following lemmas.

Proof p. 21 **Lemma 11** (Easy GLAMOUr One-Step Weak Simulation)**.** *Let s be a reachable state.*

1. Commutative & Exponential*: if $s \rightsquigarrow_{\mathtt{e},c_{1,2,3,4}} s'$ then $\underline{s} = \underline{s'}$;*
2. Multiplicative*: if $s \rightsquigarrow_{\mathtt{m}} s'$ then $\underline{s} \rightarrow_{\mathtt{r}\beta_f} \underline{s'}$.*

Proof p. 22 **Lemma 12** (Easy GLAMOUr Progress)**.** *Let s be a reachable final state. Then $\underline{s}$ is a fireball, i.e. it is β_f-normal.*

The theorem of correctness and completeness of the machine with respect to $\rightarrow_{\mathtt{r}\beta_f}$ follows. The bisimulation is *weak* because transitions other than $\rightsquigarrow_m$ are invisible on $\lambda_{\mathtt{fire}}$. For a machine execution ρ we denote with $|\rho|$ (resp. $|\rho|_{\mathtt{x}}$) the number of transitions (resp. $\mathtt{x}$-transitions for $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}, \mathtt{c}, \ldots\}$) in ρ.

Proof p. 22 **Theorem 4** (Weak Bisimulation)**.** *Let s be an initial state of code $\bar{t}$.*

1. Simulation*: For every execution $\rho : s \rightsquigarrow^* s'$ there exists a derivation d: $\underline{s} \rightarrow^*_{\mathtt{r}\beta_f} \underline{s'}$ such that $|d|_{\beta_f} = |\rho|_{\mathtt{m}}$;*
2. Reverse Simulation*: For every derivation d: $\bar{t} \rightarrow^*_{\mathtt{r}\beta_f} u$ there is an execution $\rho : s \rightsquigarrow^* s'$ such that $\underline{s'} = u$ and $|d|_{\beta_f} = |\rho|_{\mathtt{m}}$.*

Complexity Analysis The complexity analysis is divided in two parts. For any execution we show that

1. *Commutative vs Exponential*: the number of commutative transitions is (bi)linear in the number of exponential transitions and in the size of the initial term, and the cost of every commutative transition is (evidently) constant.
2. *Exponential vs Multiplicative*: the number of exponential transitions is (bi)linear in the number of multiplicative transitions and in the size of the initial term, and the cost of every exponential transition is bound by the size of the initial term.

Each point is proved via a certain measure of states and relying on an additional invariant of the machine, the *subterm invariant*. The two bounds immediately imply that the machine is linear in the number of $\rightsquigarrow_{\mathtt{m}}$ transitions—that is exactly the number of steps in the calculus, by Thm. 4—and *quadratic* in the size of the initial term.

Lemma 13 (Subterm Invariant)**.** *Let $s = (D, \bar{t}, \pi, E)$ be a state reachable from an initial code $\bar{t}_0$. If $\lambda x.\overline{u}$ is a subterm of D, $\bar{t}$, π, or E then it is a subterm of $\bar{t}_0$.* Proof p. 22

Commutative vs Exponential Transitions We define the *size* $|\bar{t}|$ of codes and the *commutative size* $|s|_c$ of states as follows:

$$|x| := 1 \qquad |\bar{t}\overline{u}| := |\bar{t}| + |\overline{u}| + 1$$
$$|\lambda x.\bar{t}| := |\bar{t}| + 1 \qquad |(D, \bar{t}, \pi, E)|_c := |\bar{t}| + \Sigma_{(\overline{u},\pi)\in D}|\overline{u}|.$$

Lemma 14 (Bilinearity of Commutative Transitions)**.** *For any state reachable by an execution ρ of initial code $\bar{t}$, $|\rho|_c \leq (1 + |\rho|_{\mathtt{e}})|\bar{t}|$.* Proof p. 22

Exponential vs Multiplicative Transitions The *free size* $|\cdot|_{\mathsf{free}}$ of a code counts the number of free variable occurrences that are not under an abstraction. It is defined recursively and extended to states as follows:

$$|x|_{\mathsf{free}} := 1 \qquad |\epsilon|_{\mathsf{free}} := 0$$
$$|\lambda y.\overline{u}|_{\mathsf{free}} := 0 \qquad |\phi : \pi|_{\mathsf{free}} := |\phi|_{\mathsf{free}} + |\pi|_{\mathsf{free}}$$
$$|\bar{t}\overline{u}|_{\mathsf{free}} := |t|_{\mathsf{free}} + |u|_{\mathsf{free}} \qquad |D : (\bar{t}, \pi)|_{\mathsf{free}} := |t|_{\mathsf{free}} + |\pi|_{\mathsf{free}} + |D|_{\mathsf{free}}$$
$$|(D, \bar{t}, \pi, E)|_{\mathsf{free}} := |D|_{\mathsf{free}} + |\bar{t}|_{\mathsf{free}} + |\pi|_{\mathsf{free}}.$$

Lemma 15 (Free Occurrences Invariant)**.** *Let $\rho : s \rightsquigarrow^* s'$ be an execution of initial code $\overline{u}$. Then $|s'|_{\mathsf{free}} \leq |\overline{u}|_{\mathsf{free}} + |\overline{u}| \cdot |\rho|_{\mathtt{m}} - |\rho|_{\mathtt{e}}$.* Proof p. 22

Corollary 4 (Exponentials are Bilinear)**.** *Let s be an initial state of code $\overline{u}$ and $\rho : s \rightsquigarrow^* s'$. Then $|\rho|_{\mathtt{e}} \leq |\overline{u}| \cdot (|\rho|_{\mathtt{m}} + 1)$.* Proof p. 23

Summing Up We can now put together the two complexity analyses, bounding the overhead of the machine.

Theorem 5 (Easy GLAMOUr Overhead Bound)**.** *Let t be a term. Every derivation $d : t \rightarrow^*_{\mathtt{r}\beta_f} u$ is implementable on RAM in $O((1 + |d|_{\beta_f}) \cdot |t|^2)$, i.e. linear in the length of d and quadratic in the size of t.* Proof p. 23

7. On the Minimality of the Cost Model

Do Inert Steps Cost 1 or 0? The number of fireball steps is a reasonable cost model for Open CBV. This roughly means that the cost of a inert step can be taken as 1, even if in Open CBV inert steps may cause size-explosion (Prop. 8). Concretely, in a reasonable implementation of Open CBV (as the Easy GLAMOUr) this is obtained by never substituting inert terms, thus handling a inert step in constant time. It is then natural to wonder if the cost of a inert step can be actually taken as 0, *i.e.* whether these inert steps can be seen as administrative work whose cost is dominated by the number of steps *by value* (or *by abstraction*), or if they are in fact computationally relevant for complexity analyses.

Here we provide evidence that the cost of a inert step is 1, not 0, *i.e.* it is relevant. Namely, we show a family of terms that evaluates in a linear number of β_λ-steps followed by an exponential number of β_i-steps. Therefore, it seems that the number of β_λ-steps is not a reasonable cost model for Open CBV. *Beware*: our family does not provide a proof that one cannot count 0, as in principle there might be an evaluation algorithm avoiding the potential exponential number of inert steps. Nonetheless, the family shows that such an algorithm, if any, is non-trivial and has to rely on some new insight to manage polynomially the exponential blow-up of inert redexes.

We build our family in two steps, first identifying a family u_n that evaluates in $O(2^n)$ β_i-steps to normal form and then building a family s_n that evaluates in $O(n)$ β_v-steps to u_n.

Step 1: Exponentially Many β_i-steps Let x, y, and z be variables and i be a inert term (that is not a variable, otherwise β_i steps will collapse on β_v steps). Consider the following three recursive families of terms (t_n and u_n are mutually recursive):

$$t_0 := x \qquad u_0 := t_0 i \qquad r_0 := u_0$$
$$t_{n+1} := \lambda z.((y u_n) u_n) \qquad u_{n+1} := t_{n+1} i \qquad r_{n+1} := (y r_n) r_n$$

Proof p. 23 **Proposition 9** (Exponentially Many β_i-Steps)**.** *For every $n \in \mathbb{N}$, one has $u_n \to_{\beta_i}^{2^n-1} r_n$.*

Step 2: Linearly Many β_v-steps Define:

$$s_0 := x \qquad s_{n+1} := (\lambda x.t_1) s_n = (\lambda x.\lambda z.(y(xi)(xi)))s_n$$

Proof p. 23 **Proposition 10** (Linearly Many β_v-Steps)**.** *For every $n \in \mathbb{N}$, one has $s_n \to_{\beta_v}^n t_n$, and so $s_n i \to_{\beta_v}^n t_n i = u_n$.*

Composing the two results, we obtain $s_n i \to_{\beta_v}^n \to_{\beta_i}^{2^n-1} r_n$, *i.e.* the family $\{s_n i\}_{n\in\mathbb{N}}$ evaluates to normal form in exponentially more β_i-steps than β_v-steps. Note that here the blow-up is also exponential with respect to the size—that is the other fundamental parameter for cost analyses—as the size of the initial term is linear in n and in the size of i, namely $|s_n i| = O(n \cdot |i|)$. Since i is arbitrary, by taking *e.g.* $i := yy$ one obtains $|s_n i| = O(n)$.

Via Thm. 1, the result transposes on $\lambda_{\mathtt{vsub}}$, giving that for a derivation d—in contrast to the fact that exponential steps are linear in the multiplicatives (*i.e.* $|d|_{\mathsf{e}} \leq |d|_{\mathsf{m}}$, Prop. 4.5)—multiplicatives may be exponential in the exponentials (*i.e.* $|d|_{\mathsf{m}} = O(2^{|d|_{\mathsf{e}}})$).

References

[1] S. Abramsky and C. L. Ong. Full Abstraction in the Lazy Lambda Calculus. *Inf. Comput.*, 105(2):159–267, 1993.

[2] B. Accattoli. Proof nets and the call-by-value λ-calculus. *Theor. Comput. Sci.*, 606:2–24, 2015.

[3] B. Accattoli and U. Dal Lago. Beta Reduction is Invariant, Indeed. In *CSL-LICS 2014*, page 8, 2014.

[4] B. Accattoli and D. Kesner. The Permutative λ-Calculus. In *LPAR*, pages 23–36, 2012.

[5] B. Accattoli and L. Paolini. Call-by-Value Solvability, revisited. In *FLOPS*, pages 4–16, 2012.

[6] B. Accattoli and C. Sacerdoti Coen. On the Value of Variables. In *WoLLIC 2014*, pages 36–50, 2014.

[7] B. Accattoli and C. Sacerdoti Coen. On the Relative Usefulness of Fireballs. Accepted at LICS 2015, 2015.

[8] B. Accattoli, P. Barenbaum, and D. Mazza. Distilling abstract machines. In *ICFP 2014*, pages 363–376, 2014.

[9] Z. M. Ariola, A. Bohannon, and A. Sabry. Sequent calculi and abstract machines. *ACM Trans. Program. Lang. Syst.*, 31(4), 2009.

[10] H. P. Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103. North-Holland, 1984.

[11] G. E. Blelloch and J. Greiner. Parallelism in Sequential Functional Languages. In *FPCA*, pages 226–237, 1995.

[12] A. Carraro and G. Guerrieri. A Semantical and Operational Account of Call-by-Value Solvability. In *FOSSACS 2014*, pages 103–118, 2014.

[13] U. Dal Lago and S. Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008.

[14] R. Dyckhoff and S. Lengrand. Call-by-value lambda-calculus and LJQ. *J. Log. Comput.*, 17(6):1109–1134, 2007.

[15] B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *(ICFP '02)*, pages 235–246, 2002.

[16] G. Guerrieri. Head reduction and normalization in a call-by-value lambda-calculus. In *WPTE 2015*, pages 3–17, 2015.

[17] G. Guerrieri, L. Paolini, and S. Ronchi Della Rocca. Standardization of a Call-By-Value Lambda-Calculus. In *TLCA 2015*, pages 211–225, 2015.

[18] H. Herbelin and S. Zimmermann. An Operational Account of Call-by-Value Minimal and Classical lambda-Calculus in "Natural Deduction" Form. In *TLCA*, pages 142–156, 2009.

[19] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. ISBN 0-13-020249-5.

[20] S. Lassen. Eager Normal Form Bisimulation. In *LICS 2005*, pages 345–354, 2005.

[21] J.-J. Lévy. Réductions correctes et optimales dans le lambda-calcul. Thése d'Etat, Univ. Paris VII, France, 1978.

[22] J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999.

[23] E. Moggi. Computational Lambda-Calculus and Monads. In *LICS '89*, pages 14–23, 1989.

[24] L. Paolini. Call-by-Value Separability and Computability. In *ICTCS*, pages 74–89, 2002.

[25] L. Paolini and S. Ronchi Della Rocca. Call-by-value Solvability. *ITA*, 33(6):507–534, 1999.

[26] L. Paolini, E. Pimentel, and S. Ronchi Della Rocca. Lazy strong normalization. In *ITRS '04*, volume 136C of *Electronic Notes in Theoretical Computer Science*, pages 103–116, 2005.

[27] L. Paolini, E. Pimentel, and S. Ronchi Della Rocca. Strong Normalization from an unusual point of view. *Theoretical Computer Science*, 412(20):1903–1915, 2011.

[28] G. D. Plotkin. Call-by-Name, Call-by-Value and the lambda-Calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[29] L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Comput. Sci.*, 2(126):281–292, 1994.

[30] S. Ronchi Della Rocca and L. Paolini. *The Parametric λ-Calculus*. Springer Berlin Heidelberg, 2004.

[31] A. Sabry and M. Felleisen. Reasoning about Programs in Continuation-Passing Style. *Lisp and Symbolic Computation*, 6(3-4):289–360, 1993.

[32] A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.

[33] D. Sands, J. Gustavsson, and A. Moran. Lambda Calculi and Linear Speedups. In *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, pages 60–84, 2002.

A. Rewriting Theory: Definitions, Notations, and Basic Results

Given a binary relation $\to_{\mathsf{r}}$ on a set I, the reflexive-transitive (resp. reflexive; transitive; reflexive-transitive and symmetric) closure of $\to_{\mathsf{r}}$ is denoted by $\to_{\mathsf{r}}^*$ (resp. $\to_{\mathsf{r}}^=$; $\to_{\mathsf{r}}^+$; $\simeq_{\mathsf{r}}$). The transpose of $\to_{\mathsf{r}}$ is denoted by ${}_{\mathsf{r}}\!\leftarrow$. A (r-)*derivation d from t to u*, denoted by $d\colon t \to_{\mathsf{r}}^* u$, is a finite sequence $(t_i)_{0\le i\le n}$ of elements of I (with $n \in \mathbb{N}$) s.t. $t = t_0$, $u = t_n$ and $t_i \to_{\mathsf{r}} t_{i+1}$ for all $1 \le i < n$;

The *number of* r*-steps of a derivation* d, *i.e.* its *length*, is denoted by $|d|_{\mathsf{r}} := n$, or simply $|d|$. If $\to_{\mathsf{r}} = \to_1 \cup \to_2$ with $\to_1 \cap \to_2 = \emptyset$, $|d|_i$ is the number of $\to_i$-steps in d, for $i = 1, 2$. We say that:

- $t \in I$ is r*-normal* or a r*-normal form* if $t \not\to_{\mathsf{r}} u$ for all $u \in I$; $u \in I$ is a r*-normal form of* t if u is r-normal and $t \to_{\mathsf{r}}^* u$;
- $t \in I$ is r*-normalizable* if there is a r-normal $u \in I$ s.t. $t \to_{\mathsf{r}}^* u$; t is *strongly* r*-normalizable* if there is no infinite sequence $(t_i)_{i\in\mathbb{N}}$ s.t. $t_0 = t$ and $t_i \to_{\mathsf{r}} t_{i+1}$;
- a r-derivation $d\colon t \to_{\mathsf{r}}^* u$ is (r-)*normalizing* if u is r-normal;
- $\to_{\mathsf{r}}$ is *strongly normalizing* if all $t \in I$ is strongly r-normalizable;
- $\to_{\mathsf{r}}$ is *strongly confluent* if, for all $t, u, s \in I$ s.t. $s\ {}_{\mathsf{r}}\!\leftarrow t \to_{\mathsf{r}} u$ and $u \neq s$, there is $r \in I$ s.t. $s \to_{\mathsf{r}} r\ {}_{\mathsf{r}}\!\leftarrow u$; $\to_{\mathsf{r}}$ is *confluent* if $\to_{\mathsf{r}}^*$ is strongly confluent.

Let $\to_1, \to_2 \subseteq I \times I$. Composition of relations is denoted by juxtaposition: for instance, $t \to_1\!\to_2 u$ means that there is $s \in I$ s.t. $t \to_1 s \to_2 u$; for any $n \in \mathbb{N}$, $t \to_1^n u$ means that there is a $\to_1$-derivation with length n ($t = u$ for $n = 0$). We say that $\to_1$ and $\to_2$ *strongly commute* if, for any $t, u, s \in I$ s.t. $u\ {}_1\!\leftarrow t \to_2 s$, one has $u \neq s$ and there is $r \in I$ s.t. $u \to_2 r\ {}_1\!\leftarrow s$. Note that if $\to_1$ and $\to_2$ strongly commute and $\to = \to_1 \cup \to_2$, then for any derivation $d\colon t \to^* u$ the sizes $|d|_1$ and $|d|_2$ are uniquely determined.

The following proposition collects some basic and well-known results of rewriting theory.

Proposition 11. *Let $\to_{\mathsf{r}}$ be a binary relation on a set I.*

1. *If $\to_{\mathsf{r}}$ is confluent then:*
   (a) *every* r*-normalizable term has a unique* r*-normal form;*
   (b) *for all $t, u \in I$, $t \simeq_{\mathsf{r}} u$ iff there is $s \in I$ s.t. $t \to_{\mathsf{r}}^* s\ {}_{\mathsf{r}}^*\!\leftarrow u$.*
2. *If $\to_{\mathsf{r}}$ is strongly confluent then $\to_{\mathsf{r}}$ is confluent and, for any $t \in I$, one has:*
   (a) *all normalizing* r*-derivations from t have the same length;*
   (b) *t is strongly* r*-normalizable if and only if t is* r*-normalizable.*

As all incarnations of Open CBV we consider are confluent, the use of Prop. 11.1 is left implicit.

For $\lambda_{\mathtt{fire}}$ and $\lambda_{\mathtt{vsub}}$, we use Prop. 11.2 and the following more informative version of Hindley–Rosen Lemma, whose proof is just a more accurate reading of the proof in [10, Prop. 3.3.5.(i)]:

Lemma 16 (Strong Hindley–Rosen). *Let $\to = \to_1 \cup \to_2$ be a binary relation on a set I s.t. $\to_1$ and $\to_2$ are strongly confluent. If $\to_1$ and $\to_2$ strongly commute, then $\to$ is strongly confluent and, for any $t \in I$ and any normalizing derivations d and e from t, one has $|d| = |e|$, $|d|_1 = |e|_1$ and $|d|_2 = |e|_2$.*

B. Omitted Proofs

B.1 Proofs of Section 2 (Incarnations of Open Call-by-Value)

***Naïve Open CBV: Plotkin's Calculus* $\lambda_{\mathtt{Plot}}$**

Remark 1. Since $\to_{\beta_v}$ does not reduce under λ's, any value is β_v-normal, and so β_y-normal and β_λ-normal, as $\to_{\beta_y}, \to_{\beta_\lambda} \subseteq \to_{\beta_v}$.

See p. 3

Proposition 1. *$\to_{\beta_y}$ $\to_{\beta_\lambda}$ and $\to_{\beta_v}$ is strongly confluent.*

Proof. We prove that $\to_{\beta_v}$ is strongly confluent. The proofs that $\to_{\beta_y}$ and $\to_{\beta_\lambda}$ are strongly confluent are perfectly analogous.

So, we prove, by induction on t, that if $t \to_{\beta_v} u$ and $t \to_{\beta_v} s$ with $u \neq s$, then there exists t' such that $u \to_{\beta_v} t'$ and $s \to_{\beta_v} t'$.

Observe that neither $t \to_{\beta_v} u$ nor $t \to_{\beta_v} s$ can be a step at the root: indeed, if $t := (\lambda x.r)v \to_{\beta_v} r\{x\leftarrow v\} =: u$ and $t \to_{\beta_v} s$ (or if $t := (\lambda x.r)v \to_{\beta_v} r\{x\leftarrow v\} =: s$ and $t \to_{\beta_v} u$), then $u = s$ since $\lambda x.r$ and v are β_v-normal by Remark 1; but this contradicts the hypothesis $u \neq s$. So, according to the definition of $t \to_{\beta_v} u$ and $t \to_{\beta_v} s$, there are only four cases.

- *Application Left for $t \to_{\beta_v} u$ and $t \to_{\beta_v} s$, i.e.* $t = rq \to_{\beta_v} pq = u$ and $t = rq \to_{\beta_v} mq = s$ with $r \to_{\beta_v} p$ and $r \to_{\beta_v} m$. By the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists r' such that $p \to_{\beta_v} r'$ and $m \to_{\beta_v} r'$. So, setting $t' = r'q$, one has $u = pq \to_{\beta_v} t'$ and $s = mq \to_{\beta_v} t'$.
- *Application Right for $t \to_{\beta_v} u$ and $t \to_{\beta_v} s$, i.e.* $t = rq \to_{\beta_v} rp = u$ and $t = rq \to_{\beta_v} rm = s$ with $q \to_{\beta_v} p$ and $q \to_{\beta_v} m$. From the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists q' such that $p \to_{\beta_v} q'$ and $m \to_{\beta_v} q'$. So, setting $t' = rq'$, one has $u = rp \to_{\beta_v} t'$ and $s = rm \to_{\beta_v} t'$.
- *Application Left for $t \to_{\beta_v} u$* and *Application Right for $t \to_{\beta_v} s$, i.e.* $t = rq \to_{\beta_v} pq = u$ and $t = rq \to_{\beta_v} rm = s$ with $r \to_{\beta_v} p$ and $q \to_{\beta_v} m$. So, setting $t' = pm$, one has $u = pq \to_{\beta_v} t'$ and $s = rm \to_{\beta_v} t'$.
- *Application Right for $t \to_{\beta_v} u$* and *Application Left for $t \to_{\beta_v} s$, i.e.* $t = rq \to_{\beta_v} rp = u$ and $t = rq \to_{\beta_v} mq = s$ with $q \to_{\beta_v} p$ and $r \to_{\beta_v} m$. So, setting $t' = mp$, one has $u = rp \to_{\beta_v} t'$ and $s = mq \to_{\beta_v} t'$. □

***Open CBV 1: the Fireball Calculus* $\lambda_{\mathtt{fire}}$**

Lemma 17 (Values and inert terms are β_f-normal)**.**

1. *Every value is β_f-normal.*
2. *Every inert term is β_f-normal.*

Proof.

1. Immediate, since $\to_{\beta_f}$ does not reduce under λ's.
2. By induction on the definition of inert term i.
   - If $i = x$ then i is obviously β_f-normal.
   - If $i = i'v$ then i' and v are β_f-normal by *i.h.* and Lemma 17.1 respectively, besides i' is not an abstraction, so i is β_f-normal.
   - Finally, if $i = i'i''$ then i' and i'' are β_f-normal by *i.h.*, moreover i' is not an abstraction, hence i is β_f-normal. □

Proposition 2 (Open Harmony)**.** *Let $t \in \Lambda$: t is β_f-normal iff t is a fireball.* See p. 3

Proof.

⇒: Proof by induction on $t \in \Lambda$. If t is a value then t is a fireball. Otherwise $t = us$ for some terms u and s. Since t is β_f-normal, then u and s are β_f-normal, and either u is not an abstraction or s is not a fireball. By induction hypothesis, u and s are fireballs. Summing up, u is either a variable or an inert term, and s is a fireball, therefore $t = us$ is an inert term and hence a fireball.

⇐: By hypothesis, t is either a value or an inert term. If t is a value, then it is β_f-normal by Lemma 17.1. Otherwise t is an inert term and then it is β_f-normal by Lemma 17.2. □

Lemma 18. *For every $t, t' \in \Lambda$, if $t \to_{\beta_i} t'$ then $t \neq t'$.*

Proof. By induction on $t \in \Lambda$. According to the definition of $t \to_{\beta_i} t'$, there are three cases.

- *Step at the root*, *i.e.* $t = (\lambda x.u)i \to_{\beta_i} u\{x \leftarrow i\} = t'$: then, since i is not an abstraction, necessarily $t = (\lambda x.u)i \neq u\{x \leftarrow i\} = t'$.
- *Application Left*, *i.e.* $t = us \to_{\beta_i} u's = t'$ with $u \to_{\beta_i} u'$: by *i.h.*, $u \neq u'$ and hence $t = us \neq u's = t'$.
- *Application Right*, *i.e.* $t = us \to_{\beta_i} us' = t'$ with $s \to_{\beta_i} s'$: by *i.h.*, $s \neq s'$ and hence $t = us \neq us' = t'$. □

See p. 3

Proposition 3 (Basic Properties of $\lambda_{\mathtt{fire}}$)**.**

1. *$\to_{\beta_i}$ is strongly normalizing and strongly confluent.*
2. *$\to_{\beta_\lambda}$ and $\to_{\beta_i}$ strongly commute.*
3. *$\to_{\beta_f}$ is strongly confluent, and all β_f-normalizing derivations d from $t \in \Lambda$ (if any) have the same length $|d|_{\beta_f}$, the same number $|d|_{\beta_\lambda}$ of β_λ-steps, and the same number $|d|_{\beta_i}$ of β_i-steps.*

Proof.

1. Strong normalization of $\to_{\beta_i}$ follows from general termination properties in the ordinary (*i.e.* pure, strong, and call-by-name) λ-calculus, as we now explain. Since β_i-steps do not substitute abstractions, they can only cause creations of type 1, according to Lévy's classification of creations of β-redexes [21]. Then β_i-derivations can be seen as special cases of m-*developments* [4], in turn a special case of more famous *superdevelopments*, *i.e.* reduction sequences reducing only (residuals of) redexes in the original term plus creations of type 1 (m-developments) or type 1 and 2 (superdevelopments). Both m-developments and superdevelopments always terminate [4]. Therefore, $\to_{\beta_i}$ is strongly normalizing.

   Now, we prove that $\to_{\beta_i}$ is strongly confluent, that is if $t \to_{\beta_i} u$ and $t \to_{\beta_i} s$ with $u \neq s$, then there exists $t' \in \Lambda$ such that $u \to_{\beta_i} t'$ and $s \to_{\beta_i} t'$. The proof is by induction on $t \in \Lambda$. Observe that neither $t \to_{\beta_i} u$ nor $t \to_{\beta_i} s$ can be a step at the root: indeed, if $t := (\lambda x.r)i \mapsto_{\beta_i} r\{x \leftarrow i\} := u$ and $t \to_{\beta_i} s$ (or if $t := (\lambda x.r)i \mapsto_{\beta_i} r\{x \leftarrow i\} =: s$ and $t \to_{\beta_i} u$), then $u = s$ since $\lambda x.r$ and i are β_i-normal by Lemmas 17.1-2 (as $\to_{\beta_i} \subseteq \to_{\beta_f}$); but this contradicts the hypothesis $u \neq s$. So, according to the definition of $t \to_{\beta_i} u$ and $t \to_{\beta_i} s$, there are only four cases.
   - *Application Left for $t \to_{\beta_i} u$ and $t \to_{\beta_i} s$, i.e. $t = rq \to_{\beta_i} pq = u$ and $t = rq \to_{\beta_i} mq = s$ with $r \to_{\beta_i} p$ and $r \to_{\beta_i} m$.* By the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists r' such that $p \to_{\beta_i} r'$ and $m \to_{\beta_i} r'$. So, setting $t' = r'q$, one has $u = pq \to_{\beta_i} t'$ and $s = mq \to_{\beta_i} t'$.
   - *Application Right for $t \to_{\beta_i} u$ and $t \to_{\beta_i} s$, i.e. $t = rq \to_{\beta_i} rp = u$ and $t = rq \to_{\beta_i} rm = s$ with $q \to_{\beta_i} p$ and $q \to_{\beta_i} m$.* By the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists q' such that $p \to_{\beta_i} q'$ and $m \to_{\beta_i} q'$. So, setting $t' = rq'$, one has $u = rp \to_{\beta_i} t'$ and $s = rm \to_{\beta_i} t'$.
   - *Application Left for $t \to_{\beta_i} u$* and *Application Right for $t \to_{\beta_i} s$, i.e. $t = rq \to_{\beta_i} pq = u$ and $t = rq \to_{\beta_i} rm = s$* with $r \to_{\beta_i} p$ and $q \to_{\beta_i} m$. So, setting $t' = pm$, one has $u = pq \to_{\beta_i} t'$ and $s = rm \to_{\beta_i} t'$.
   - *Application Right for $t \to_{\beta_i} u$* and *Application Left for $t \to_{\beta_i} s$, i.e. $t = rq \to_{\beta_i} rp = u$ and $t = rq \to_{\beta_i} mq = s$* with $q \to_{\beta_i} p$ and $r \to_{\beta_i} m$. So, setting $t' = mp$, one has $u = rp \to_{\beta_i} t'$ and $s = mq \to_{\beta_i} t'$.
2. We prove, by induction on $t \in \Lambda$, that if $t \to_{\beta_\lambda} u$ and $t \to_{\beta_i} s$, then $u \neq s$ and there is $t' \in \Lambda$ such that $u \to_{\beta_i} t'$ and $s \to_{\beta_\lambda} t'$. Observe that neither $t \to_{\beta_\lambda} u$ nor $t \to_{\beta_i} s$ can be a step at the root: indeed, if $t := (\lambda x.r)\lambda y.q \mapsto_{\beta_\lambda} r\{x \leftarrow \lambda y.q\} =: u$ (resp. $t := (\lambda x.r)i \mapsto_{\beta_i} r\{x \leftarrow i\} =: s$) then $\lambda y.q$ is not a inert term (resp. i is not an abstraction), moreover $\lambda x.r$ and $\lambda y.q$ (resp. i) are β_i-normal (resp. β_λ-normal) by Prop. 2, as $\to_{\beta_i} \subseteq \to_{\beta_f}$ (resp. $\to_{\beta_\lambda} \subseteq \to_{\beta_f}$); therefore, t is β_i-normal (resp. β_λ-normal) but this contradicts the hypothesis $t \to_{\beta_i} s$ (resp. $t \to_{\beta_\lambda} u$). So, according to the definitions of $t \to_{\beta_\lambda} u$ and $t \to_{\beta_i} s$, there are only four cases.
   - *Application Left for both $t \to_{\beta_\lambda} u$ and $t \to_{\beta_i} s$, i.e. $t := rq \to_{\beta_\lambda} pq =: u$* and $t := rq \to_{\beta_i} mq =: s$ with $r \to_{\beta_\lambda} p$ and $r \to_{\beta_i} m$. By *i.h.*, $p \neq m$ and there exists r' such that $p \to_{\beta_i} r'$ and $m \to_{\beta_\lambda} r'$. So, $u \neq s$ and, setting $t' := r'q$, one has $u = pq \to_{\beta_i} t'$ $_{\beta_\lambda}\!\leftarrow mq = s$.
   - *Application Right for both $t \to_{\beta_\lambda} u$ and $t \to_{\beta_i} s$, i.e. $t := rq \to_{\beta_\lambda} rp =: u$* and $t := rq \to_{\beta_i} rm =: s$ with $q \to_{\beta_\lambda} p$ and $q \to_{\beta_i} m$. By *i.h.*, $p \neq m$ and there exists q' such that $p \to_{\beta_i} q'$ and $m \to_{\beta_\lambda} q'$. So, $u \neq s$ and, setting $t' := rq'$, one has $u = rp \to_{\beta_i} t'$ $_{\beta_\lambda}\!\leftarrow rm = s$.
   - *Application Left for $t \to_{\beta_\lambda} u$* and *Application Right for $t \to_{\beta_i} s$, i.e. $t := rq \to_{\beta_\lambda} pq = u$* and $t = rq \to_{\beta_i} rm =: s$ with $r \to_{\beta_\lambda} p$ and $q \to_{\beta_i} m$. By Lemma 18, $q \neq m$ and hence $u = pq \neq rm = s$. Setting $t' := pm$, one has $u = pq \to_{\beta_i} t'$ $_{\beta_\lambda}\!\leftarrow rm = s$.
   - *Application Right for $t \to_{\beta_\lambda} u$* and *Application Left for $t \to_{\beta_i} s$, i.e. $t := rq \to_{\beta_\lambda} rp =: u$* and $t = rq \to_{\beta_i} mq = s$ with $q \to_{\beta_\lambda} p$ and $r \to_{\beta_i} m$. By Lemma 18, $r \neq m$ and hence $u = rp \neq mq = s$. Setting $t' := mp$, one has $u = rp \to_{\beta_i} t'$ $_{\beta_\lambda}\!\leftarrow mq = s$.
3. It follows immediately from strong confluence of $\to_{\beta_\lambda}$ (Prop. 1.1) and $\to_{\beta_i}$ (Prop. 3.1), the strong commutation of $\to_{\beta_\lambda}$ and $\to_{\beta_i}$ (Prop. 3.2), and Hindley-Rosen (Lemma 16). □

Open CBV 2: the Value Substitution Calculus $\lambda_{\mathtt{vsub}}$

Proposition 4 (Basic Properties of $\lambda_{\mathtt{vsub}}$, [5])**.** See p. 4

1. *$\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are strongly normalizing (separately).*
2. *$\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are strongly confluent (separately).*
3. *$\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ strongly commute.*
4. *$\to_{\mathtt{vsub}}$ is strongly confluent, and all $\mathtt{vsub}$-normalizing derivations d from $t \in \Lambda_{\mathtt{vsub}}$ (if any) have the same length $|d|_{\mathtt{vsub}}$, the same number $|d|_{\mathsf{e}}$ of e-steps, and the same number $|d|_{\mathsf{e}}$ of m-steps.*
5. *Let $t \in \Lambda$. For any $\mathtt{vsub}$-derivation d from t, $|d|_{\mathsf{e}} \leq |d|_{\mathsf{m}}$.*

Proof. The statements of Prop. 4 are a refinement of some results proved in [5], where $\to_{\mathtt{vsub}}$ is denoted by $\to_{\mathtt{w}}$.

1. In [5, Lemma 3] it has been proved that $\to_{\mathtt{dB}}$ and $\to_{\mathtt{vs}}$ are strongly normalizing, separately. Since $\to_{\mathsf{m}} \subseteq \to_{\mathtt{dB}}$ and $\to_{\mathsf{e}} \subseteq \to_{\mathtt{vs}}$ ($\to_{\mathtt{dB}}$ and $\to_{\mathtt{vs}}$ are just the extensions of $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$, respectively, obtained by allowing reductions under λ's), one has that $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are strongly normalizing, separately.
2. We prove that $\to_{\mathsf{m}}$ is strongly confluent, *i.e.* if u $_{\mathsf{m}}\!\leftarrow t \to_{\mathsf{m}} s$ with $u \neq s$ then there exists $t' \in \Lambda_{\mathtt{vsub}}$ such that $u \to_{\mathsf{m}} t'$ $_{\mathsf{m}}\!\leftarrow s$. The proof is by induction on the definition of $\to_{\mathsf{m}}$. Since there $t \to_{\mathsf{m}} s \neq u$ and the reduction $\to_{\mathsf{m}}$ is weak, there are only eight cases:
   - *Step at the Root for $t \to_{\mathsf{m}} u$ and Application Right for $t \to_{\mathsf{m}} s$, i.e. $t := L\langle \lambda x.q\rangle r \mapsto_{\mathsf{m}} L\langle q[x \leftarrow r]\rangle =: u$* and $t \mapsto_{\mathsf{m}} L\langle \lambda x.q\rangle r' =: s$ with $r \to_{\mathsf{m}} r'$: then, $u \to_{\mathsf{m}} L\langle q[x \leftarrow r']\rangle$ $_{\mathsf{m}}\!\leftarrow s$;
   - *Step at the Root for $t \to_{\mathsf{m}} u$ and Application Left for $t \to_{\mathsf{m}} s$, i.e.*, for some $n > 0$, $t := (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]r \mapsto_{\mathsf{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: u$ whereas $t \to_{\mathsf{m}} (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]r =: s$ with $t_j \to_{\mathsf{m}} t'_j$ for some $1 \leq j \leq n$: then,

$$u \to_{\mathsf{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] \ {}_{\mathsf{m}}\!\leftarrow s;$$

- *Application Left for* $t \to_{\mathsf{m}} u$ *and Application Right for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq rq \to_{\mathsf{m}} r'q =: u$ and $t \to_{\mathsf{m}} rq' =: s$ with $r \to_{\mathsf{m}} r'$ and $q \to_{\mathsf{m}} q'$: then, $u \to_{\mathsf{m}} r'q' \,{}_{\mathsf{m}}\!\leftarrow s$;
- *Application Left for both* $t \to_{\mathsf{m}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq rq \to_{\mathsf{m}} r'q =: u$ and $t \to_{\mathsf{m}} r''q =: s$ with $r' \,{}_{\mathsf{m}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} r_0 \,{}_{\mathsf{m}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} r_0 q \,{}_{\mathsf{m}}\!\leftarrow s$;
- *Application Right for both* $t \to_{\mathsf{m}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq qr \to_{\mathsf{m}} qr' =: u$ and $t \to_{\mathsf{m}} qr'' =: s$ with $r' \,{}_{\mathsf{m}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} r_0 \,{}_{\mathsf{m}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} qr_0 \,{}_{\mathsf{m}}\!\leftarrow s$;
- *ES Left for* $t \to_{\mathsf{m}} u$ *and ES Right for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq r[x\leftarrow q] \to_{\mathsf{m}} r'[x\leftarrow q] =: u$ and $t \to_{\mathsf{m}} r[x\leftarrow q'] =: s$ with $r \to_{\mathsf{m}} r'$ and $q \to_{\mathsf{m}} q'$: then, $u \to_{\mathsf{m}} r'[x\leftarrow q'] \,{}_{\mathsf{m}}\!\leftarrow s$;
- *ES Left for both* $t \to_{\mathsf{m}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq r[x\leftarrow q] \to_{\mathsf{m}} r'[x\leftarrow q] =: u$ and $t \to_{\mathsf{m}} r''[x\leftarrow q] =: s$ with $r' \,{}_{\mathsf{m}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} r_0 \,{}_{\mathsf{m}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} r_0[x\leftarrow q] \,{}_{\mathsf{m}}\!\leftarrow s$;
- *ES Right for both* $t \to_{\mathsf{m}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq q[x\leftarrow r] \to_{\mathsf{m}} q[x\leftarrow r'] =: u$ and $t \to_{\mathsf{m}} q[x\leftarrow r''] =: s$ with $r' \,{}_{\mathsf{m}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} r_0 \,{}_{\mathsf{m}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} q[x\leftarrow r_0] \,{}_{\mathsf{m}}\!\leftarrow s$.

We prove that $\to_{\mathsf{e}}$ is strongly confluent, *i.e.* if $u \,{}_{\mathsf{e}}\!\leftarrow t \to_{\mathsf{e}} s$ with $u \neq s$ then there exists $r \in \Lambda_{\mathtt{vsub}}$ such that $u \to_{\mathsf{e}} t' \,{}_{\mathsf{e}}\!\leftarrow s$. The proof is by induction on the definition of $\to_{\mathsf{e}}$. Since there $t \to_{\mathsf{e}} s \neq u$ and the reduction $\to_{\mathsf{e}}$ is weak, there are only eight cases:

- *Step at the Root for* $t \to_{\mathsf{e}} u$ and *ES Left for* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq r[x\leftarrow L\langle v\rangle] \mapsto_{\mathsf{e}} L\langle r\{x\leftarrow v\}\rangle =: u$ and $t \mapsto_{\mathsf{e}} r'[x\leftarrow L\langle v\rangle] =: s$ with $r \to_{\mathsf{e}} r'$: then, $u \to_{\mathsf{e}} L\langle r'[x\leftarrow v]\rangle \,{}_{\mathsf{e}}\!\leftarrow s$;
- *Step at the Root for* $t \to_{\mathsf{e}} u$ and *ES Right for* $t \to_{\mathsf{e}} s$, *i.e.*, for some $n > 0$, $t \coloneqq r[x\leftarrow v[x_1\leftarrow t_1]\dots[x_n\leftarrow t_n]] \mapsto_{\mathsf{e}} r\{x\leftarrow v\}[x_1\leftarrow t_1]\dots[x_n\leftarrow t_n] =: u$ whereas $t \to_{\mathsf{e}} r[x\leftarrow v[x_1\leftarrow t_1]\dots[x_j\leftarrow t'_j]\dots[x_n\leftarrow t_n]] =: s$ with $t_j \to_{\mathsf{e}} t'_j$ for some $1 \le j \le n$: then,

$$u \to_{\mathsf{e}} r\{x\leftarrow v\}[x_1\leftarrow t_1]\dots[x_j\leftarrow t'_j]\dots[x_n\leftarrow t_n] \,{}_{\mathsf{e}}\!\leftarrow s;$$

- *Application Left for* $t \to_{\mathsf{e}} u$ and *Application Right for* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq rq \to_{\mathsf{e}} r'q =: u$ and $t \to_{\mathsf{e}} rq' =: s$ with $r \to_{\mathsf{e}} r'$ and $q \to_{\mathsf{e}} q'$: then, $u \to_{\mathsf{e}} r'q' \,{}_{\mathsf{e}}\!\leftarrow s$;
- *Application Left for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq rq \to_{\mathsf{e}} r'q =: u$ and $t \to_{\mathsf{e}} r''q =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{e}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{e}} r_0 \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{e}} r_0 q \,{}_{\mathsf{e}}\!\leftarrow s$;
- *Application Right for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq qr \to_{\mathsf{e}} qr' =: u$ and $t \to_{\mathsf{e}} qr'' =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{e}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{e}} r_0 \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{e}} qr_0 \,{}_{\mathsf{e}}\!\leftarrow s$;
- *ES Left for* $t \to_{\mathsf{e}} u$ and *ES Right for* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq r[x\leftarrow q] \to_{\mathsf{e}} r'[x\leftarrow q] =: u$ and $t \to_{\mathsf{e}} r[x\leftarrow q'] =: s$ with $r \to_{\mathsf{e}} r'$ and $q \to_{\mathsf{e}} q'$: then, $u \to_{\mathsf{e}} r'[x\leftarrow q'] \,{}_{\mathsf{e}}\!\leftarrow s$;
- *ES Left for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq r[x\leftarrow q] \to_{\mathsf{e}} r'[x\leftarrow q] =: u$ and $t \to_{\mathsf{e}} r''[x\leftarrow q] =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{e}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{e}} r_0 \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{e}} r_0[x\leftarrow q] \,{}_{\mathsf{e}}\!\leftarrow s$;
- *ES Right for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{e}} s$, *i.e.* $t \coloneqq q[x\leftarrow r] \to_{\mathsf{e}} q[x\leftarrow r'] =: u$ and $t \to_{\mathsf{e}} q[x\leftarrow r''] =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{e}} r''$: by *i.h.*, there exists $r_0 \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{e}} r_0 \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{e}} q[x\leftarrow r_0] \,{}_{\mathsf{e}}\!\leftarrow s$.

Note that in [5, Lemma 11] it has just been proved the strong confluence of $\to_{\mathtt{vsub}}$, not of $\to_{\mathsf{m}}$ or $\to_{\mathsf{e}}$.

3. We show that $\to_{\mathsf{e}}$ and $\to_{\mathsf{m}}$ strongly commute, *i.e.* if $u \,{}_{\mathsf{e}}\!\leftarrow t \to_{\mathsf{m}} s$, then $u \neq s$ and there is $t' \in \Lambda_{\mathtt{vsub}}$ such that $u \to_{\mathsf{m}} t' \,{}_{\mathsf{e}}\!\leftarrow s$.

   The proof is by induction on the definition of $t \to_{\mathsf{e}} u$. The proof that $u \neq s$ is left to the reader. Since the $\to_{\mathsf{e}}$ and $\to_{\mathsf{m}}$ cannot reduce under λ's, all $\mathtt{vsub}$-values are m-normal and e-normal. So, there are the following cases.

   - *Step at the Root for* $t \to_{\mathsf{e}} u$ and *ES Left for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq r[z\leftarrow L\langle v\rangle] \to_{\mathsf{e}} L\langle r\{z\leftarrow v\}\rangle =: u$ and $t \to_{\mathsf{m}} r'[z\leftarrow L\langle v\rangle] =: s$ with $r \to_{\mathsf{m}} r'$: then $u \to_{\mathsf{m}} L\langle r'\{z\leftarrow v\}\rangle \,{}_{\mathsf{e}}\!\leftarrow u$;
   - *Step at the Root for* $t \to_{\mathsf{e}} u$ and *ES Right for* $t \to_{\mathsf{m}} s$, *i.e.*

$$\begin{aligned} t &\coloneqq r[z\leftarrow v[x_1\leftarrow t_1]\dots[x_n\leftarrow t_n]] \\ &\to_{\mathsf{e}} r\{z\leftarrow v\}[x_1\leftarrow t_1]\dots[x_n\leftarrow t_n] =: u \end{aligned}$$

     and $t \to_{\mathsf{m}} r[z\leftarrow v[x_1\leftarrow t_1]\dots[x_j\leftarrow t'_j]\dots[x_n\leftarrow t_n]] =: s$ for some $n > 0$, and $t_j \to_{\mathsf{m}} t'_j$ for some $1 \le j \le n$: then, $u \to_{\mathsf{m}} r\{z\leftarrow v\}[x_1\leftarrow t_1]\dots[x_j\leftarrow t'_j]\dots[x_n\leftarrow t_n] \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *Application Left for* $t \to_{\mathsf{e}} u$ and *Application Right for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq rq \to_{\mathsf{e}} r'q =: u$ and $t \to_{\mathsf{m}} rq' =: s$ with $r \to_{\mathsf{e}} r'$ and $q \to_{\mathsf{m}} q'$: then, $t \to_{\mathsf{m}} r'q' \,{}_{\mathsf{e}}\!\leftarrow u$;
   - *Application Left for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq rq \to_{\mathsf{e}} r'q =: u$ and $t \to_{\mathsf{m}} r''q =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*, there exists $p \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} p \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} pq \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *Application Left for* $t \to_{\mathsf{e}} u$ and *Step at the Root for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq (\lambda x.q)[x_1\leftarrow t_1]\dots[x_n\leftarrow t_n]r \to_{\mathsf{e}} (\lambda x.q)[x_1\leftarrow t_1]\dots[x_j\leftarrow t'_j]\dots[x_n\leftarrow t_n]r =: u$ with $n > 0$ and $t_j \to_{\mathsf{e}} t'_j$ for some $1 \le j \le n$, and $t \to_{\mathsf{m}} q[x\leftarrow r][x_1\leftarrow t_1]\dots[x_n\leftarrow t_n] =: s$: then,

$$u \to_{\mathsf{m}} q[x\leftarrow r][x_1\leftarrow t_1]\dots[x_j\leftarrow t'_j]\dots[x_n\leftarrow t_n] \,{}_{\mathsf{e}}\!\leftarrow s;$$

   - *Application Right for* $t \to_{\mathsf{e}} u$ and *Application Left for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq qr \to_{\mathsf{e}} qr' =: u$ and $t \to_{\mathsf{m}} q'r =: s$ with $r \to_{\mathsf{e}} r'$ and $q \to_{\mathsf{m}} q'$: then, $u \to_{\mathsf{m}} q'r' \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *Application Right for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq qr \to_{\mathsf{e}} qr' =: u$ and $t \to_{\mathsf{m}} qr'' =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*, there exists $p \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} p \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} qp \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *Application Right for* $t \to_{\mathsf{e}} u$ and *Step at the Root for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq L\langle\lambda x.q\rangle r \to_{\mathsf{e}} L\langle\lambda x.q\rangle r' =: u$ with $r \to_{\mathsf{e}} r'$, and $t \to_{\mathsf{m}} L\langle q[x\leftarrow r]\rangle =: s$: then, $u \to_{\mathsf{m}} L\langle q[x\leftarrow r']\rangle \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *ES Left for* $t \to_{\mathsf{e}} u$ and *ES Right for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq r[x\leftarrow q] \to_{\mathsf{e}} r'[x\leftarrow q] =: u$ and $t \to_{\mathsf{m}} r[x\leftarrow q'] =: s$ with $r \to_{\mathsf{e}} r'$ and $q \to_{\mathsf{m}} q'$: then, $u \to_{\mathsf{m}} r'[x\leftarrow q'] \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *ES Left for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq r[x\leftarrow q] \to_{\mathsf{e}} r'[x\leftarrow q] =: u$ and $t \to_{\mathsf{m}} r''[x\leftarrow q] =: s$ with $r' \,{}_{\mathsf{e}}\!\leftarrow r \to_{\mathsf{m}} r''$: by *i.h.*,there exists $p \in \Lambda_{\mathtt{vsub}}$ such that $r' \to_{\mathsf{m}} p \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} p[x\leftarrow q] \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *ES Right for* $t \to_{\mathsf{e}} u$ and *ES Left for* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq q[x\leftarrow r] \to_{\mathsf{e}} q[x\leftarrow r'] =: u$ and $t \to_{\mathsf{m}} q'[x\leftarrow r] =: s$ with $r \to_{\mathsf{e}} r'$ and $q \to_{\mathsf{m}} q'$: then, $u \to_{\mathsf{m}} q'[x\leftarrow r'] \,{}_{\mathsf{e}}\!\leftarrow s$;
   - *ES Right for both* $t \to_{\mathsf{e}} u$ *and* $t \to_{\mathsf{m}} s$, *i.e.* $t \coloneqq q[x\leftarrow r] \to_{\mathsf{e}} q[x\leftarrow r'] =: u$ and $t \to_{\mathsf{m}} q[x\leftarrow r''] =: s$ with $r \,{}_{\mathsf{e}}\!\leftarrow r' \to_{\mathsf{m}} r''$: by *i.h.*, there exists $p \in \Lambda_{\mathtt{vsub}}$ such that $r \to_{\mathsf{m}} p \,{}_{\mathsf{e}}\!\leftarrow r''$, hence $u \to_{\mathsf{m}} q[x\leftarrow p] \,{}_{\mathsf{e}}\!\leftarrow s$.

4. It follows immediately from strong confluence of $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ (Prop. 4.2), strong commutation of $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ (Prop. 4.3) and Hindley-Rosen (Lemma 16).
   A different proof of the strong confluence of $\to_{\mathtt{vsub}}$ (without information about the number of steps) is in [5, Lemma 11].
5. The intuition behind the proof is that any m-step creates a new ES, any e-step erases an ES. Formally, let $u \in \Lambda_{\mathtt{vsub}}$ such that $d\colon t \to^*_{\mathtt{vsub}} u$. We prove by induction on $|d|_{\mathtt{vsub}} \in \mathbb{N}$ that $|d|_{\mathsf{e}} = |d|_{\mathsf{m}} - |u|_{\mathrm{ES}}$ (where $|u|_{\mathrm{ES}}$ is the number of ES in u) and any $\mathtt{vsub}$-value that is a subterm of u is a value (without ES).
   If $|d|_{\mathtt{vsub}} = 0$, then $u = t \in \Lambda$, then we can conclude.

Suppose $|d|_{\mathtt{vsub}} > 0$: then, d is the concatenation of $d' : t \to^*_{\mathtt{vsub}} s$ and $s \to_{\mathtt{vsub}} u$, for some $s \in \Lambda_{\mathtt{vsub}}$. By *i.h.*, $|d'|_{\mathsf{e}} = |d'|_{\mathsf{m}} - |s|_{\mathrm{ES}}$ and that every vsub-value that is a subterm of s is a value (without ES). There are two cases:

- $s := E\langle r[x{\leftarrow}L\langle v\rangle]\rangle \to_{\mathsf{e}} E\langle L\langle r\{x{\leftarrow}v\}\rangle\rangle =: u$, then $|d|_{\mathsf{m}} = |d'|_{\mathsf{m}}$ and $|s|_{\mathrm{ES}} = |u|_{\mathrm{ES}} + 1$, since $|v|_{\mathrm{ES}} = 0$ by *i.h.*; therefore $|d|_{\mathsf{e}} = |d'|_{\mathsf{e}} + 1 = |d'|_{\mathsf{m}} - |s|_{\mathrm{ES}} + 1 = |d|_{\mathsf{m}} - |u|_{\mathrm{ES}}$ and any vsub-value that is a subterm of u is a value (without ES).
- $s := E\langle L\langle \lambda x.r\rangle q\rangle \to_{\mathsf{m}} E\langle L\langle r[x{\leftarrow}q]\rangle\rangle =: u$, then $|u|_{\mathrm{ES}} = |s|_{\mathrm{ES}} + 1$ and $|d|_{\mathsf{m}} = |d'|_{\mathsf{m}} + 1$, therefore $|d|_{\mathsf{e}} = |d'|_{\mathsf{e}} = |d'|_{\mathsf{m}} - |s|_{\mathrm{ES}} = |d|_{\mathsf{m}} - |u|_{\mathrm{ES}}$. Moreover, the new occurrence of ES $[x{\leftarrow}q]$ in u cannot be under the scope of a λ, otherwise the redex in s which is fired in the m-step would be under the scope of a λ, but this is impossible since $\to_{\mathsf{m}}$ is a weak reduction. So, any vsub-value that is a subterm of u is a value (without ES).

□

Open CBV 3: the Shuffling Calculus $\lambda_{\mathtt{sh}}$

Definition 1 (Occurrences)**.** *For all $t \in \Lambda$, let $[t]_\lambda$ be the number of occurrences of λ in t, and $[t]_x$ be the number of free occurrences of the variable x in t, and $\mathsf{sub}_u(t)$ be the number of occurrences in t of the term u.*

Remark 2. Since $\to_{\beta^\flat_v}$ and $\to_{\sigma^\flat}$ do not reduce under λ's without argument, every value is $\beta^\flat_v$-normal and $\sigma^\flat$-normal, and hence sh-normal.

Remark 3. The reduction $\to_{\sigma^\flat}$ is just the closure under balanced contexts of the binary relation $\mapsto_\sigma = \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3}$ on Λ (see definitions in Fig. 4).

Lemma 19. *Let $t, t' \in \Lambda$.*

1. *For every value v, if $t \to_{\sigma^\flat} t'$ then $(\lambda x.t')v \neq t\{x{\leftarrow}v\}$.*
2. *If $t \to_{\sigma^\flat} t'$ then $t \neq t'$.*
3. *For every value v, one has $t\{x{\leftarrow}v\} \neq \lambda x.tv$.*

Proof.

1. By induction on the definition of $t \to_{\sigma^\flat} t'$, using Remark 3.
2. In [12, Proposition 2] it has been proved that there exists a size $\#\colon \Lambda \to \mathbb{N}$ such that if $t \to_\sigma t'$ then $\#(t) > \#(t')$, where $\to_\sigma$ is just the extension of $\to_{\sigma^\flat}$ obtained by allowing reductions under λ's. Therefore, $\to_{\sigma^\flat} \subseteq \to_\sigma$ and hence if $t \to_{\sigma^\flat} t'$ then $\#(t) > \#(t')$, in particular $t \neq t'$.
3. According to Definition 1, $[t\{x{\leftarrow}v\}]_\lambda = [t]_\lambda + [v]_\lambda \cdot [t]_x$ and $[\lambda x.tv]_\lambda = 1 + [t]_\lambda + [v]_\lambda$, and $[t\{x{\leftarrow}v\}]_x = [t]_x \cdot [v]_x$ and $[\lambda x.tv]_x = 0$.
   Suppose $t\{x{\leftarrow}v\} = \lambda x.tv$: then, $[t\{x{\leftarrow}v\}]_\lambda = [\lambda x.tv]_\lambda$ and $[t\{x{\leftarrow}v\}]_x = [\lambda x.tv]_x$, thus
   $$[v]_\lambda \cdot [t]_x = 1 + [v]_\lambda \qquad [t]_x \cdot [v]_x = 0. \tag{3}$$
   The only solution to the first equation of (3) is $[v]_\lambda = 1$ and $[t]_x = 2$, whence $[v]_x = 0$ according to the second equation of (3). As $x \notin \mathtt{fv}(v)$, one has $\mathsf{sub}_v(\lambda x.tv) = 1 + \mathsf{sub}_v(t)$ and $\mathsf{sub}_v(t\{x{\leftarrow}v\}) = \mathsf{sub}_v(t) + [t]_x = \mathsf{sub}_v(t) + 2$, therefore $\mathsf{sub}_v(\lambda x.tv) \neq \mathsf{sub}_v(t\{x{\leftarrow}v\})$ and hence $\lambda x.tv \neq t\{x{\leftarrow}v\}$. Contradiction. □

See p. 5

Proposition 5 (Basic Properties of $\lambda_{\mathtt{sh}}$, [12])**.**

1. *Let $t, u, s \in \Lambda$: if $t \to_{\beta^\flat_v} u$ and $t \to_{\sigma^\flat} s$ then $u \neq s$.*
2. *$\to_{\sigma^\flat}$ is strongly normalizing and (not strongly) confluent.*
3. *$\to_{\mathtt{sh}}$ is (not strongly) confluent.*
4. *Let $t \in \Lambda$: t is strongly sh-normalizable iff t is sh-normalizable.*

Proof.

1. By induction on $t \in \Lambda$. According to the definition of $t \to_{\sigma^\flat} s$ and Remark 3, the following cases are impossible.
   - *Step at the root for $t \to_{\beta^\flat_v} u$* and either the *Step at the root* or the *Application Left* or the *Application Right for $t \to_{\sigma^\flat} s$*. Indeed, if $t = (\lambda x.r)v \mapsto_{\beta_v} r\{x{\leftarrow}v\} = u$ then $\lambda x.r$ and v are $\sigma^\flat$-normal by Remark 2; moreover t is neither a σ_1-redex nor a σ_3-redex, because $\lambda x.r$ and v, respectively, are not applications.
   - *Application Left for $t \to_{\beta^\flat_v} u$* and *Step inside a β-context for $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} pq = u$* with $r \to_{\beta^\flat_v} p$, and $t = (\lambda x.r')q \to_{\sigma^\flat} (\lambda x.m)q = s$ with $r = \lambda x.r'$ and $r' \to_{\sigma^\flat} m$. Indeed r is $\beta^\flat_v$-normal by Remark 2.
   - *Step inside a β-context for $t \to_{\beta^\flat_v} u$* and *Application Left for $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\sigma^\flat} pq = s$* with $r \to_{\sigma^\flat} p$, and $t = (\lambda x.r')q \to_{\beta^\flat_v} (\lambda x.m)q = u$ with $r = \lambda x.r'$ and $r' \to_{\beta^\flat_v} m$. Indeed r is $\sigma^\flat$-normal by Remark 2.

   Therefore, according to the definition of $t \to_{\sigma^\flat} s$ and Remark 3, there are "only" eleven cases.
   - *Step at the root for $t \to_{\beta^\flat_v} u$* and *Step inside a β-context for $t \to_{\sigma^\flat} s$, i.e. $t = (\lambda x.r)v \mapsto_{\beta_v} r\{x{\leftarrow}v\} = u$* and $t = (\lambda x.r)v \to_{\sigma^\flat} (\lambda x.r')v = s$ with $r \to_{\sigma^\flat} r'$. By Lemma 19.1, $u \neq s$.
   - *Application Left for $t \to_{\beta^\flat_v} u$* and *Step at the root for $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} pq = u$* with $r \to_{\beta^\flat_v} p$, and $t \mapsto_\sigma s$ (see Remark 3). It is impossible that $t \mapsto_{\sigma_3} s$, otherwise r would be a value and hence $\beta^\flat_v$-normal by Remark 2, but this contradicts that $r \to_{\beta^\flat_v} p$. Thus, $t = (\lambda x.r')r''q \mapsto_{\sigma_1} (\lambda x.r'q)r'' = s$ with $x \notin \mathtt{fv}(q)$ and $r = (\lambda x.r')r''$. We claim that $u \neq s$. Indeed, if $u = s$ then $q = r''$ and $p = \lambda x.r'q$ with $r = (\lambda x.r')q \to_{\beta^\flat_v} \lambda x.r'q = p$, hence necessarily $r \mapsto_{\beta_v} p$ (*i.e.* $r \to_{\beta^\flat_v} p$ by a step at the root) and thus q is a value and $\lambda x.r'q = p = r'\{x{\leftarrow}q\}$, but this is impossible by Lemma 19.3.
   - *Application Left for $t \to_{\beta^\flat_v} u$ and $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} pq = u$* and $t = rq \to_{\sigma^\flat} mq = s$ with $r \to_{\beta^\flat_v} p$ and $r \to_{\sigma^\flat} m$. By *i.h.*, $p \neq m$ and hence $u = pq \neq mq = s$.
   - *Application Left for $t \to_{\beta^\flat_v} u$* and *Application Right for $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} pq = u$* and $t = rq \to_{\sigma^\flat} rm = s$, with $r \to_{\beta^\flat_v} p$ and $q \to_{\sigma^\flat} m$. By Lemma 19.2, $q \neq m$ and hence $u = pq \neq rm = s$.
   - *Application Right for $t \to_{\beta^\flat_v} u$* and *Step at the root for $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} rp = u$* with $q \to_{\beta^\flat_v} p$, and $t \mapsto_\sigma s$ (see Remark 3).
     - If $t \mapsto_{\sigma_1} s$ then $t = (\lambda x.r')r''q \mapsto_{\sigma_1} (\lambda x.r'q)r'' = s$ with $x \notin \mathtt{fv}(q)$ and $r = (\lambda x.r')r''$. We claim that $u \neq s$. Indeed, if $u = s$ then $p = r''$ and $r = \lambda x.r'q$, therefore $(\lambda x.r')p = r = \lambda x.r'q$ which is impossible.
     - If $t \mapsto_{\sigma_3} s$ then $t = r((\lambda x.q')q'') \mapsto_{\sigma_3} (\lambda x.rq')q'' = s$ where r is a value, $x \notin \mathtt{fv}(r)$ and $q = (\lambda x.q')q''$. We claim that $u \neq s$. Indeed, if $u = s$ then $r = \lambda x.rq'$ which is impossible.
   - *Application Right for $t \to_{\beta^\flat_v} u$ and $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} pq = u$* and $t = rq \to_{\sigma^\flat} mq = s$ with $q \to_{\beta^\flat_v} p$ and $q \to_{\sigma^\flat} m$. By *i.h.*, $p \neq m$ and hence $u = rp \neq rm = s$.
   - *Application Right for $t \to_{\beta^\flat_v} u$* and *Application Left for $t \to_{\sigma^\flat} s$, i.e. $t = rq \to_{\beta^\flat_v} rp = u$* and $t = rq \to_{\sigma^\flat} mq = s$, with $q \to_{\beta^\flat_v} p$ and $r \to_{\sigma^\flat} m$. By Lemma 19.2, $r \neq m$ and hence $u = rp \neq mq = s$.

- *Application Right for* $t \to_{\beta_v^\flat} u$ and *Step inside a* β*-context for* $t \to_{\sigma^\flat} s$, *i.e.* $t = rq \to_{\beta_v^\flat} rp = u$ with $q \to_{\beta_v^\flat} p$, and $t = (\lambda x.r')q \to_{\sigma^\flat} (\lambda x.m)q = s$ with $r = \lambda x.r'$ and $r' \to_{\sigma^\flat} m$. By Lemma 19.2, $r' \neq m$ whence $r = \lambda x.r' \neq \lambda x.m$ and thus $u \neq s$.
- *Step inside a* β*-context for* $t \to_{\beta_v^\flat} u$ and *Step at the root for* $t \to_{\sigma^\flat} s$, *i.e.* $t = (\lambda x.r)q \to_{\beta_v^\flat} (\lambda x.r')q = u$ with $r \to_{\beta_v^\flat} r'$, and $t \mapsto_\sigma s$ (see Remark 3). It is impossible that $t = (\lambda x.r)q \mapsto_{\sigma_1} s$ because $\lambda x.r$ is not an application. Thus, $t = (\lambda x.r)((\lambda y.q')q'') \mapsto_{\sigma_3} (\lambda y.(\lambda x.r)q')q'' = s$ with $q = (\lambda y.q')q''$ and $y \notin \mathtt{fv}(\lambda x.r)$, therefore $q \neq q''$ and hence $u \neq s$.
- *Step inside a* β*-context for* $t \to_{\beta_v^\flat} u$ and *Application Right for* $t \to_{\sigma^\flat} s$, *i.e.* $t = rq \to_{\sigma^\flat} rp = s$ with $q \to_{\sigma^\flat} p$, and $t = (\lambda x.r')q \to_{\beta_v^\flat} (\lambda x.m)q = u$ with $r = \lambda x.r'$ and $r' \to_{\beta_v^\flat} m$. By Lemma 19.2, $q \neq p$ whence $u \neq s$.
- *Step inside a* β*-context for* $t \to_{\beta_v^\flat} u$ *and* $t \to_{\sigma^\flat} s$, *i.e.* $t = (\lambda x.r)q \to_{\beta_v} (\lambda x.p)q = u$ and $t = (\lambda x.r)q \to_{\sigma^\flat} (\lambda x.m)q = s$ with $r \to_{\beta_v^\flat} p$ and $r \to_{\sigma^\flat} m$. By *i.h.*, $p \neq m$ and hence $u \neq s$.

2. In [12, Proposition 2] it has been proved that $\to_\sigma$ is strongly normalizing, where $\to_\sigma$ is just the extension of $\to_{\sigma^\flat}$ obtained by allowing reductions under λ's. Therefore, $\to_{\sigma^\flat} \subseteq \to_\sigma$ and hence $\to_{\sigma^\flat}$ is strongly normalizing.
   The (not strong) confluence of $\to_{\sigma^\flat}$ has been proved in [12, Lemma 9.ii], where $\to_{\sigma^\flat}$ is denoted by $\to_{\mathtt{w}[\sigma]}$.
3. See [12, Proposition 10], where $\to_{\mathtt{sh}}$ is denoted by $\to_{\mathtt{w}}$.
4. See [12, Theorem 24], where $\to_{\mathtt{sh}}$ is denoted by $\to_{\mathtt{w}}$. □

B.2 Proofs of Section 3 (Quantitative Termination Equivalences)

Simulating $\lambda_{\mathtt{fire}}$ ***in*** $\lambda_{\mathtt{vsub}}$

Remark 4. Let $t, u \in \Lambda_{\mathtt{vsub}}$.

1. If $t \equiv u$ then $t{\downarrow} = u{\downarrow}$.
2. If $t \equiv u$ then $t \not\to_{\mathtt{vsub}} u$ (in particular, $t \not\to_{\mathtt{m}} u$ and $t \not\to_{\mathtt{e}} u$).

See p. 6

Lemma 2 (Simulation of a $\to_{\beta_f}$-Step by $\to_{\mathtt{vsub}}$)**.** *Let* $t, u \in \Lambda$.

1. *If* $t \to_{\beta_\lambda} u$ *then* $t \to_{\mathtt{m}} \to_{\mathtt{e}} u$.
2. *If* $t \to_{\beta_i} u$ *then* $t \to_{\mathtt{m}} \equiv s$, *with* $s \in \Lambda_{\mathtt{vsub}}$ *clean and* $s{\downarrow} = u$.

Proof. Both proofs are by induction on the rewriting step.

1. According to the definition of $t \to_{\beta_\lambda} u$, there are three cases:
   - *Step at the root*, *i.e.* $t = (\lambda x.s)(\lambda y.r) \mapsto_{\beta_\lambda} s\{x \leftarrow \lambda y.r\} = u$: so, $t \to_{\mathtt{m}} s[x \leftarrow \lambda y.r] \to_{\mathtt{e}} u$.
   - *Application Left*, *i.e.* $t = sr \to_{\beta_\lambda} s'r = u$ with $s \to_{\beta_\lambda} s'$: by *i.h.*, $s \to_{\mathtt{m}} \to_{\mathtt{e}} s'$ and hence $t = sr \to_{\mathtt{m}} \to_{\mathtt{e}} s'r = u$.
   - *Application Right*, *i.e.* $t = sr \to_{\beta_\lambda} sr' = u$ with $r \to_{\beta_\lambda} r'$: by *i.h.*, $r \to_{\mathtt{m}} \to_{\mathtt{e}} r'$ and hence $t = sr \to_{\mathtt{m}} \to_{\mathtt{e}} sr' = u$.
2. According to the definition of $t \to_{\beta_i} u$, there are three cases:
   - *Step at the root*, *i.e.* $t = (\lambda x.s)i \mapsto_{\beta_i} s\{x \leftarrow i\} = u$: then, $t \to_{\mathtt{m}} s[x \leftarrow i]$ where $s[x \leftarrow i]$ is clean (since $s \in \Lambda$) and $s[x \leftarrow i]{\downarrow} = s{\downarrow}\{x \leftarrow i{\downarrow}\} = u$ ($s{\downarrow} = s$ and $i{\downarrow} = i$ because $s, i \in \Lambda$). We conclude since $\equiv$ is reflexive.
   - *Application Left*, *i.e.* $t = sr \to_{\beta_i} s'r = u$ with $s \to_{\beta_i} s'$: by *i.h.*, $s \to_{\mathtt{m}} \equiv q$ where q is a clean vsub-term such that $q{\downarrow} = s'$. So, $q = q_0[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$ where $q_0 \in \Lambda$ and $i_1, \dots, i_n$ are inert terms (for some $n \in \mathbb{N}$), moreover we can suppose without loss of generality that $\{x_1, \dots, x_n\} \cap \mathtt{fv}(r) = \emptyset$. Let $u' = (q_0 r)[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$: then, u' is a clean vsub-term such that $qr \equiv u'$ and, according to Remark 4.1, $u'{\downarrow} = (qr){\downarrow} = q{\downarrow}r{\downarrow} = s'r = u$. Hence, $t = sr \to_{\mathtt{m}} \equiv qr \equiv u'$ and we conclude since $\equiv$ is transitive.
   - *Application Right*, *i.e.* $t = sr \to_{\beta_i} sr' = u$ with $r \to_{\beta_i} r'$. Identical to the *application left* case, just switch left and right. □

Lemma 20 (Fireballs are Closed Under Anti-Substitution of Inert Terms)**.** *Let* t *be a* vsub*-term and* i *be an inert term.*

1. *If* $t\{x \leftarrow i\}$ *is an abstraction then* t *is an abstraction.*
2. *If* $t\{x \leftarrow i\}$ *is an inert term then* t *is an inert term;*
3. *If* $t\{x \leftarrow i\}$ *is a fireball then* t *is a fireball.*

Proof.

1. If $t\{x \leftarrow i\} = \lambda y.s$ then there is r such that $s = r\{x \leftarrow i\}$, that is $t\{x \leftarrow i\} = \lambda y.(r\{x \leftarrow i\}) = (\lambda y.r)\{x \leftarrow i\}$ and so $t = \lambda y.r$ is an abstraction;
2. By induction on the inert structure of $t\{x \leftarrow i\}$. Cases:
   - *Variable*, *i.e.* $t\{x \leftarrow i\} = y$, possibly with $x = y$. Then $t = x$ or $t = y$, and in both cases t is inert.
   - *Compound Inert*, *i.e.* $t\{x \leftarrow i\} = i'f$. If t is a variable then it is inert. Otherwise it is an application $t = us$, and so $u\{x \leftarrow i\} = i'$ and $s\{x \leftarrow i\} = f$. By *i.h.*, u is an inert term. Consider f. Two cases:
     (a) f is an abstraction. Then by Point 1 s is an abstraction.
     (b) f is an inert term. Then by *i.h.* s is an inert term.
     In both cases s is a fireball, and so $t = us$ is a inert term.
3. Immediate consequence of Lemmas 20.1-2, since every fireball is either an abstraction or an inert term. □

Lemma 21 (Substitution of Inert Terms Does Not Create β_f-Redexes)**.** *Let* t, u *be terms and* i *be an inert term. There is* $s \in \Lambda$ *such that:*

1. *if* $t\{x \leftarrow i\} \to_{\beta_\lambda} u$ *then* $t \to_{\beta_\lambda} s$ *and* $s\{x \leftarrow i\} = u$;
2. *if* $t\{x \leftarrow i\} \to_{\beta_i} u$ *then* $t \to_{\beta_i} s$ *and* $s\{x \leftarrow i\} = u$.

Proof. We prove the two points by induction on the evaluation context closing the root redex. Cases:

- *Step at the root*:
  1. *Abstraction Step*, *i.e.* $t\{x \leftarrow i\} := (\lambda y.r\{x \leftarrow i\})q\{x \leftarrow i\} \mapsto_{\beta_\lambda} r\{x \leftarrow i\}\{y \leftarrow q\{x \leftarrow i\}\} =: u$. By Lemma 20.1, q is an abstraction, since $q\{x \leftarrow i\}$ is an abstraction by hypothesis. Then $t = (\lambda y.r)q \mapsto_{\beta_\lambda} r\{y \leftarrow q\}$. Then $s := r\{x \leftarrow q\}$ verifies the statement, as $s\{x \leftarrow i\} = (r\{y \leftarrow q\})\{x \leftarrow i\} = r\{x \leftarrow i\}\{y \leftarrow q\{x \leftarrow i\}\} = u$.
  2. *Inert Step*, identical to the abstraction subcase, just replace *abstraction* with *inert term* and the use of Lemma 20.1 with the use of Lemma 20.2.
- *Application Left*, *i.e.* $t = rq$ and reduction takes place in r:
  1. *Abstraction Step*, *i.e.* $t\{x \leftarrow i\} := r\{x \leftarrow i\}q\{x \leftarrow i\} \to_{\beta_\lambda} pq\{x \leftarrow i\} =: u$. By *i.h.* there exists $s' \in \Lambda$ such that $p = s'\{x \leftarrow i\}$ and $r \to_{\beta_\lambda} s'$. Then $s := s'q$ satisfies the statement, as $s\{x \leftarrow i\} = (s'q)\{x \leftarrow i\} = s'\{x \leftarrow i\}q\{x \leftarrow i\} = u$.
  2. *Inert Step*, identical to the abstraction subcase.
- *Application Right*, *i.e.* $t = rq$ and reduction takes place in q. Identical to the *application left* case, just switch left and right. □

Lemma 3 (Projection of a β_f-Step on $\to_{\mathtt{vsub}}$ via Unfolding)**.** *Let* t *be a clean* vsub*-term and* u *be a term.* See p. 6

1. *If* $t{\downarrow} \to_{\beta_\lambda} u$ *then* $t \to_{\mathtt{m}} \to_{\mathtt{e}} s$, *with* $s \in \Lambda_{\mathtt{vsub}}$ *clean s.t.* $s{\downarrow} = u$.
2. *If* $t{\downarrow} \to_{\beta_i} u$ *then* $t \to_{\mathtt{m}} \equiv s$, *with* $s \in \Lambda_{\mathtt{vsub}}$ *clean s.t.* $s{\downarrow} = u$.

Proof. Since t is clean, there are a λ-term q and some inert λ-terms $i_1, \dots, i_n$ (with $n \in \mathbb{N}$) such that $t = q[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$. We prove both points by induction on $n \in \mathbb{N}$. The base case (*i.e.* $n = 0$) is given by the simulation of one-step reductions given by Lemma 2, since $t = q \in \Lambda$ and hence $t{\downarrow} = t$ (recall that, when applying Lemma 2.1, $u \in \Lambda$ implies that u is clean and $u{\downarrow} = u$).

Consider now $n > 0$. Let $t_{n-1} := q[x_1 \leftarrow i_1] \dots [x_{n-1} \leftarrow i_{n-1}]$: so, $t = t_{n-1}[x_n \leftarrow i_n]$ and $t{\downarrow} = t_{n-1}{\downarrow}\{x_n \leftarrow i_n\}$. Both points rely on the fact that the substitution of inert terms cannot create redexes (Lemma 21). Namely,

1. β_λ*-step*: the application of Lemma 21.1 to $t{\downarrow} = t_{n-1}{\downarrow}\{x_n \leftarrow i_n\} \to_{\beta_\lambda} u$ (since $t_{n-1}{\downarrow} \in \Lambda$, *i.e.* it has no ES) provides $r \in \Lambda$ such that $t_{n-1}{\downarrow} \to_{\beta_\lambda} r$ and $r\{x_n \leftarrow i_n\} = u$. By *i.h.*, $t_n \to_{\mathsf{m}} \to_{\mathsf{e}} s$ where s is a clean $\mathtt{vsub}$-term such that $s{\downarrow} = r$, and thus $t = t_{n-1}[x_n \leftarrow i_n] \to_{\mathsf{m}} \to_{\mathsf{e}} s[x_n \leftarrow i_n]$. Moreover, $s[x_n \leftarrow i_n]$ is clean and $s[x_n \leftarrow i_n]{\downarrow} = s{\downarrow}\{x_n \leftarrow i_n\} = r\{x_n \leftarrow i_n\} = u$.
2. β_i*-step*: the application of Lemma 21.2 to $t{\downarrow} = t_{n-1}{\downarrow}\{x_n \leftarrow i_n\} \to_{\beta_i} u$ provides $r \in \Lambda$ such that $t_{n-1}{\downarrow} \to_{\beta_i} r$ and $r\{x_n \leftarrow i_n\} = u$. By *i.h.*, $t_{n-1} \to_{\mathsf{m}} \equiv s$ where s is a clean $\mathtt{vsub}$-term such that $s{\downarrow} = r$; thus, $t = t_{n-1}[x_n \leftarrow i_n] \to_{\mathsf{m}} \equiv s[x_n \leftarrow i_n]$. Moreover, $s[x_n \leftarrow i_n]$ is clean and $s[x_n \leftarrow i_n]{\downarrow} = s{\downarrow}\{x_n \leftarrow i_n\} = r\{x_n \leftarrow i_n\} = u$. □

See p. 6 **Lemma 4.** *Let t be a clean $\mathtt{vsub}$-term. If $t{\downarrow}$ is a fireball, then t is $\{\mathtt{m}, \mathtt{e}_\lambda\}$-normal and its body is a fireball.*

Proof. First, we prove that if $t{\downarrow}$ is a fireball then for some fireball f and inert terms $i_1, \dots i_n$ one has $t = f[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$. Since t is clean, there are a λ-term u and some inerts λ-terms $i_1, \dots, i_n$ (with $n \in \mathbb{N}$) such that $t = u[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$. We prove by induction on $n \in \mathbb{N}$ that u is a fireball.

If $n = 0$, then $t = u \in \Lambda$, thus $u = t{\downarrow}$ and hence u is a fireball.

Suppose $n > 0$ and let $s := u[x_1 \leftarrow i_1] \dots [x_{n-1} \leftarrow i_{n-1}]$, which is a clean $\mathtt{vsub}$-term: then, $t = s[x_n \leftarrow i_n]$ and hence $t{\downarrow} = s{\downarrow}\{x_n \leftarrow i_n\}$ (as $i_n{\downarrow} = i_n$ because $i_n \in \Lambda$). By Lemma 20.3, $s{\downarrow}$ is a fireball. By *i.h.*, u is a fireball.

Now, fireballs are $\mathtt{vsub}$-normal. Indeed, a fireball is without ES, hence it is without e-redexes, moreover it is immediate to prove that fireballs are m-normal (by simply adapting the proof of Lemma 17).

So, $t = f[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$ can only have e_y-redexes. □

See p. 6 **Lemma 5** (Linear Postponement of $\to_{\mathsf{e}_y}$)**.** *Let $t, u, s \in \Lambda_{\mathtt{vsub}}$.*

1. *If $t \to_{\mathsf{e}_y} s \to_{\mathsf{m}} u$ then $t \to_{\mathsf{m}} \to_{\mathsf{e}_y} u$.*
2. *If $t \to_{\mathsf{e}_y} \to_{\mathsf{e}_\lambda} u$ then $t \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} u$.*
3. *If $d \colon t \to^*_{\mathtt{vsub}} u$ then $e \colon t \to^*_{\mathsf{m},\mathsf{e}_\lambda} \to^*_{\mathsf{e}_y} u$ with $|e|_{\mathtt{vsub}} = |d|_{\mathtt{vsub}}$, $|e|_{\mathsf{m}} = |d|_{\mathsf{m}}$, $|e|_{\mathsf{e}} = |d|_{\mathsf{e}}$, and $|e|_{\mathsf{e}_\lambda} \geq |d|_{\mathsf{e}_\lambda}$.*

Proof. 1. By induction on the definition of $t \to_{\mathsf{e}_y} s$. Since the e_y-step cannot create in s new m-redexes not occurring in t, the m-redex fired in $s \to_{\mathsf{m}} u$ is (a residual of a m-redex) already occurring in t. So, there are the following cases.

- *Step at the Root for $t \to_{\mathsf{e}_y} s$ and ES Left for $s \to_{\mathsf{m}} u$, i.e. $t := r[z \leftarrow L\langle x\rangle] \to_{\mathsf{e}_y} L\langle r\{z \leftarrow x\}\rangle =: s$ and $s \to_{\mathsf{m}} L\langle r'\{z \leftarrow x\}\rangle =: u$ with $r \to_{\mathsf{m}} r'$*: then $t \to_{\mathsf{m}} r'[z \leftarrow L\langle x\rangle] \to_{\mathsf{e}_y} u$;
- *Step at the Root for $t \to_{\mathsf{e}_y} s$ and ES "quasi-Right" for $s \to_{\mathsf{m}} u$, i.e. $t := r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \to_{\mathsf{e}_y} r\{z \leftarrow x\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: s$* and

$$t \to_{\mathsf{m}} r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] =: u$$

for some $n > 0$, and $t_j \to_{\mathsf{m}} t'_j$ for some $1 \leq j \leq n$: then, $t \to_{\mathsf{m}} r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] \to_{\mathsf{e}_y} u$;

- *Application Left for $t \to_{\mathsf{e}_y} s$* and *Application Right for $s \to_{\mathsf{m}} u$, i.e. $t := rq \to_{\mathsf{e}_y} r'q =: s$* and *$s \to_{\mathsf{m}} r'q' =: u$* with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{m}} q'$: then, $t \to_{\mathsf{m}} rq' \to_{\mathsf{e}_y} u$;
- *Application Left for both $t \to_{\mathsf{e}_y} s$ and $s \to_{\mathsf{m}} u$, i.e. $t := rq \to_{\mathsf{e}_y} r'q =: s$* and *$s \to_{\mathsf{m}} r''q =: u$* with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{m}} r''$: by *i.h.*, $r \to_{\mathsf{m}} \to_{\mathsf{e}_y} r''$, hence $t \to_{\mathsf{m}} \to_{\mathsf{e}_y} u$;
- *Application Left for $t \to_{\mathsf{e}_y} s$* and *Step at the Root for $s \to_{\mathsf{m}} u$, i.e. $t := (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]r \to_{\mathsf{e}_y} (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]r =: s$* with $n > 0$ and $t_j \to_{\mathsf{e}_y} t'_j$ for some $1 \leq j \leq n$, and *$s \to_{\mathsf{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] =: u$*: then,

$$t \to_{\mathsf{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] \to_{\mathsf{e}_y} u;$$

- *Application Right for $t \to_{\mathsf{e}_y} s$* and *Application Left for $s \to_{\mathsf{m}} u$, i.e. $t := qr \to_{\mathsf{e}_y} qr' =: s$* and *$s \to_{\mathsf{m}} q'r' =: u$* with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{m}} q'$: then, $t \to_{\mathsf{m}} q'r \to_{\mathsf{e}_y} u$;
- *Application Right for both $t \to_{\mathsf{e}_y} s$ and $s \to_{\mathsf{m}} u$, i.e. $t := qr \to_{\mathsf{e}_y} qr' =: s$* and *$s \to_{\mathsf{m}} qr'' =: u$* with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{m}} r''$: by *i.h.*, $r \to_{\mathsf{m}} \to_{\mathsf{e}_y} r''$, hence $t \to_{\mathsf{m}} \to_{\mathsf{e}_y} u$;
- *Application Right for $t \to_{\mathsf{e}_y} s$* and *Step at the Root for $s \to_{\mathsf{m}} u$, i.e. $t := L\lambda x.qr \to_{\mathsf{e}_y} L\langle \lambda x.q\rangle r' =: s$* with $r \to_{\mathsf{e}_y} r'$, and *$s \to_{\mathsf{m}} L\langle q[x \leftarrow r']\rangle =: u$*: then, $t \to_{\mathsf{m}} L\langle q[x \leftarrow r]\rangle \to_{\mathsf{e}_y} u$;
- *ES Left for $t \to_{\mathsf{e}_y} s$* and *ES Right for $s \to_{\mathsf{m}} u$, i.e. $t := r[x \leftarrow q] \to_{\mathsf{e}_y} r'[x \leftarrow q] =: s$* and *$s \to_{\mathsf{m}} r'[x \leftarrow q'] =: u$* with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{m}} q'$: then, $t \to_{\mathsf{m}} r[x \leftarrow q'] \to_{\mathsf{e}_y} u$;
- *ES Left for both $t \to_{\mathsf{e}_y} s$ and $s \to_{\mathsf{m}} u$, i.e. $t := r[x \leftarrow q] \to_{\mathsf{e}_y} r'[x \leftarrow q] =: s$* and *$s \to_{\mathsf{m}} r''[x \leftarrow q] =: u$* with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{m}} r''$: by *i.h.*, $r \to_{\mathsf{m}} \to_{\mathsf{e}_y} r''$, hence $t \to_{\mathsf{m}} \to_{\mathsf{e}_y} u$;
- *ES Right for $t \to_{\mathsf{e}_y} s$* and *ES Left for $s \to_{\mathsf{m}} u$, i.e. $t := q[x \leftarrow r] \to_{\mathsf{e}_y} q[x \leftarrow r'] =: s$* and *$s \to_{\mathsf{m}} q'[x \leftarrow r'] =: u$* with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{m}} q'$: then, $t \to_{\mathsf{m}} q'[x \leftarrow r] \to_{\mathsf{e}_y} u$;
- *ES Right for both $t \to_{\mathsf{e}_y} s$ and $s \to_{\mathsf{m}} u$, i.e. $t := q[x \leftarrow r] \to_{\mathsf{e}_y} q[x \leftarrow r'] =: s$* and *$s \to_{\mathsf{m}} q[x \leftarrow r''] =: u$* with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{m}} r''$: by *i.h.*, $r \to_{\mathsf{m}} \to_{\mathsf{e}_y} r''$, hence $t \to_{\mathsf{m}} \to_{\mathsf{e}_y} u$.

2. By induction on the definition of $t \to_{\mathsf{e}_y} s$. Since the e_y-step cannot create in s new e_λ-redexes not occurring in t, the e_λ-redex fired in $s \to_{\mathsf{e}_\lambda} u$ is (a residual of a e_λ-redex) already occurring in t. So, there are the following cases.

- *Step at the Root for both $t \to_{\mathsf{e}_y} s$ and $s \to_{\mathsf{e}_\lambda} u$, i.e. $t := r[x \leftarrow L'\langle z\rangle[y \leftarrow L\langle \lambda x.q\rangle]] \to_{\mathsf{e}_y} L'\langle r\{x \leftarrow z\}\rangle[y \leftarrow L\langle \lambda x.q\rangle] =: s$* and *$s \to_{\mathsf{e}_\lambda} L\langle L'\langle r\{x \leftarrow z\}\rangle\{y \leftarrow \lambda x.q\}\rangle =: u$* (with possibly $y = z$). We set $L'' := L'\{y \leftarrow \lambda x.q\}$ *i.e.* L'' is the substitution context obtained from L' by the capture-avoiding substitution of $\lambda x.q$ for each free occurrence of y in L'. We can suppose without loss of generality that $y \notin \mathtt{fv}(L) \cup \mathtt{fv}(r)$. There are two sub-cases:
  - either $y = z$ and then $t \to_{\mathsf{e}_\lambda} r[x \leftarrow L\langle L''\langle \lambda x.q\rangle\rangle] \to_{\mathsf{e}_\lambda} L\langle L''\langle r\{x \leftarrow \lambda x.q\}\rangle\rangle = u$,
  - or $y \neq z$ and then $t \to_{\mathsf{e}_\lambda} r[x \leftarrow L\langle L''\langle z\rangle\rangle] \to_{\mathsf{e}_y} L\langle L''\langle r\{x \leftarrow z\}\rangle\rangle = u$.
- *Step at the Root for $t \to_{\mathsf{e}_y} s$ and ES Left for $s \to_{\mathsf{e}_\lambda} u$, i.e. $t := r[z \leftarrow L\langle x\rangle] \to_{\mathsf{e}_y} L\langle r\{z \leftarrow x\}\rangle =: s$* and *$s \to_{\mathsf{e}_\lambda} L\langle r'\{z \leftarrow x\}\rangle =: u$* with $r \to_{\mathsf{e}_\lambda} r'$: then $t \to_{\mathsf{e}_\lambda} r'[z \leftarrow L\langle x\rangle] \to_{\mathsf{e}_y} u$;
- *Step at the Root for $t \to_{\mathsf{e}_y} s$ and ES "quasi-Right" for $s \to_{\mathsf{e}_\lambda} u$, i.e. $t := r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \to_{\mathsf{e}_y} r\{z \leftarrow x\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: s$* for some $n > 0$, and $t_j \to_{\mathsf{e}_\lambda} t'_j$ for some $1 \leq j \leq n$, and *$s \to_{\mathsf{e}_\lambda} r\{z \leftarrow x\}[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] =: u$*: then, $t \to_{\mathsf{e}_\lambda} r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] \to_{\mathsf{e}_y} u$;

- *Application Left for* $t \to_{\mathsf{e}_y} s$ *and Application Right for* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := rq \to_{\mathsf{e}_y} r'q =: s$ and $s \to_{\mathsf{e}_\lambda} r'q' =: u$ with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{e}_\lambda} q'$: then, $t \to_{\mathsf{e}_\lambda} rq' \to_{\mathsf{e}_y} u$;
- *Application Left for both* $t \to_{\mathsf{e}_y} s$ *and* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := rq \to_{\mathsf{e}_y} r'q =: s$ and $s \to_{\mathsf{e}_\lambda} r''q =: u$ with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{e}_\lambda} r''$: by *i.h.*, $r \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} r''$, hence $t \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} u$;
- *Application Right for* $t \to_{\mathsf{e}_y} s$ *and Application Left for* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := qr \to_{\mathsf{e}_y} qr' =: s$ and $s \to_{\mathsf{e}_\lambda} q'r' =: u$ with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{e}_\lambda} q'$: then, $t \to_{\mathsf{e}_\lambda} q'r \to_{\mathsf{e}_y} u$;
- *Application Right for both* $t \to_{\mathsf{e}_y} s$ *and* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := qr \to_{\mathsf{e}_y} qr' =: s$ and $s \to_{\mathsf{e}_\lambda} qr'' =: u$ with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{e}_\lambda} r''$: by *i.h.*, $r \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} r''$, hence $t \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} u$;
- *ES Left for* $t \to_{\mathsf{e}_y} s$ *and Step at the Root for* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := r[z \leftarrow L\langle \lambda y.q\rangle] \to_{\mathsf{e}_y} r'[z \leftarrow L\langle \lambda y.q\rangle] =: s$ and $s \to_{\mathsf{e}_\lambda} L\langle r'\{z \leftarrow \lambda y.q\}\rangle =: u$ with $r \to_{\mathsf{e}_y} r'$: this means that in r there is an ES of the form $[y \leftarrow x]$ (possibly $x = z$) which is fired in $r \to_{\mathsf{e}_y} r'$; then, $t \to_{\mathsf{e}_\lambda} L\langle r\{z \leftarrow \lambda y.q\}\rangle \to_{\mathsf{e}} u$, where the last e-step is a e_λ-step if $x = z$, otherwise it is a e_y-step;
- *ES Left for* $t \to_{\mathsf{e}_y} s$ *and ES Right for* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := r[x \leftarrow q] \to_{\mathsf{e}_y} r'[x \leftarrow q] =: s$ and $s \to_{\mathsf{e}_\lambda} r'[x \leftarrow q'] =: u$ with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{e}_\lambda} q'$: then, $t \to_{\mathsf{e}_\lambda} r[x \leftarrow q'] \to_{\mathsf{e}_y} u$;
- *ES Left for both* $t \to_{\mathsf{e}_y} s$ *and* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := r[x \leftarrow q] \to_{\mathsf{e}_y} r'[x \leftarrow q] =: s$ and $s \to_{\mathsf{e}_\lambda} r''[x \leftarrow q] =: u$ with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{e}_\lambda} r''$: by *i.h.*, $r \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} r''$, so $t \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} u$;
- *ES Right for* $t \to_{\mathsf{e}_y} s$ *and Step at the Root for* $s \to_{\mathsf{e}_\lambda} u$, *i.e.*

  $$t := r[z \leftarrow (\lambda y.q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]]$$
  $$\to_{\mathsf{e}_y} r[z \leftarrow (\lambda y.q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] =: s$$

  for some $n > 0$, and $t_j \to_{\mathsf{e}_y} t'_j$ for some $1 \le j \le n$, and $s \to_{\mathsf{e}_\lambda} r\{z \leftarrow \lambda y.q\}[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] =: u$: then, $t \to_{\mathsf{e}_\lambda} r\{z \leftarrow \lambda y.q\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] \to_{\mathsf{e}_y} u$;
- *ES Right for* $t \to_{\mathsf{e}_y} s$ *and ES Left for* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := q[x \leftarrow r] \to_{\mathsf{e}_y} q[x \leftarrow r'] =: s$ and $s \to_{\mathsf{e}_\lambda} q'[x \leftarrow r'] =: u$ with $r \to_{\mathsf{e}_y} r'$ and $q \to_{\mathsf{e}_\lambda} q'$: then, $t \to_{\mathsf{e}_\lambda} q'[x \leftarrow r] \to_{\mathsf{e}_y} u$;
- *ES Right for both* $t \to_{\mathsf{e}_y} s$ *and* $s \to_{\mathsf{e}_\lambda} u$, *i.e.* $t := q[x \leftarrow r] \to_{\mathsf{e}_y} q[x \leftarrow r'] =: s$ and $s \to_{\mathsf{e}_\lambda} q[x \leftarrow r''] =: u$ with $r \to_{\mathsf{e}_y} r'$ and $r' \to_{\mathsf{e}_\lambda} r''$: by *i.h.*, $r \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} r''$, hence $t \to_{\mathsf{e}_\lambda} \to_{\mathsf{e}} u$.

3. By induction on $|d|_{\mathtt{vsub}} \in \mathbb{N}$, using Lemmas 5.1-2 in the inductive case. □

See p. 6

Theorem 1 (Quantitative Simulation of $\lambda_{\mathtt{fire}}$ in $\lambda_{\mathtt{vsub}}$). *Let* $t, u \in \Lambda$. *If* $d\colon t \to^*_{\beta_f} u$ *then there are* $s, r \in \Lambda_{\mathtt{vsub}}$ *and* $e\colon t \to^*_{\mathtt{vsub}} r$ *such that*

1. Qualitative Relationship*:* $r \equiv s$, $u = s{\downarrow} = r{\downarrow}$ *and* s *is clean;*
2. Quantitative Relationship*:*
   (a) Multiplicative Steps: $|d|_{\beta_f} = |e|_{\mathsf{m}}$;
   (b) Exponential (Abstraction) Steps: $|d|_{\beta_\lambda} = |e|_{\mathsf{e}_\lambda} = |e|_{\mathsf{e}}$.
3. Normal Forms*: if* u *is* β_f*-normal then there exists* $f\colon r \to^*_{\mathsf{e}_y} q$ *such that* q *is a* vsub*-normal form and* $|f|_{\mathsf{e}_y} \le |e|_{\mathsf{m}} - |e|_{\mathsf{e}_\lambda}$.

Proof. The first two points are proved together.

1-2. By the remark at the beginning of this section of the Appendix (Remarks 4.1-2), it is sufficient to show that there exists $e\colon t \to^*_{\mathtt{vsub}} \equiv s \in \Lambda_{\mathtt{vsub}}$ such that $u = s{\downarrow}$ with s clean, and $|d|_{\beta_f} = |e|_{\mathsf{m}}$ and $|d|_{\beta_\lambda} = |e|_{\mathsf{e}_\lambda}$ (the fact that $|e|_{\mathsf{e}_\lambda} = |e|_{\mathsf{e}}$ is immediate, since the simulation obtained by iterating the projection in Lemma 3 never uses $\to_{\mathsf{e}_y}$). We proceed by induction on $|d|_{\beta_f} \in \mathbb{N}$. Cases:

- *Empty derivation*, *i.e.* $|d|_{\beta_f} = 0$ then $t = u$ and $|d|_{\beta_\lambda} = 0$, so we conclude taking $s := u$ and e as the empty derivation.
- *Non-empty derivation*, *i.e.* $|d|_{\beta_f} > 0$: then, $d\colon t \to^*_{\beta_f} r \to_{\beta_f} u$ and let $d'\colon t \to^*_{\beta_f} r$ be the derivation obtained from d by removing its last step $r \to_{\beta_f} u$. By *i.h.*, there is $e'\colon t \to^*_{\mathtt{vsub}} \equiv q$ such that $r = q{\downarrow}$, q is clean, $|d'|_{\beta_f} = |e'|_{\mathsf{m}}$, and $|d'|_{\beta_\lambda} = |e'|_{\mathsf{e}_\lambda}$. By applying Lemma 3 to the last step $r \to_{\beta_f} u$ of d, we obtain s such that either $q \to_{\mathsf{m}} \to_{\mathsf{e}} s$, if $q{\downarrow} \to_{\beta_v} u$, or $q \to_{\mathsf{m}} \equiv s$, if $q{\downarrow} \to_{\beta_i} u$, and in both cases s is a clean vsub-term such that $s'{\downarrow} = u$. Note that both cases can be summed up with $q \to_{\mathsf{m}} \overset{\equiv}{\to}_{\mathsf{e}} s$. Composing the two obtained derivations $e'\colon t \to^*_{\mathtt{vsub}} \equiv q$ and $q \to_{\mathsf{m}} \overset{\equiv}{\to}_{\mathsf{e}} s$, we obtain the derivation $e''\colon t \to^*_{\mathtt{vsub}} \equiv q \to_{\mathsf{m}} \overset{\equiv}{\to}_{\mathsf{e}} s$ that satisfies the quantitative relationships but not yet the qualitative one, as $\equiv$ appears between two steps of e''. It is then enough to apply the strong bisimulation property of $\equiv$ (Lemma 1.2), that provides a derivation $e : t \to^*_{\mathtt{vsub}} \to_{\mathsf{m}} \overset{\equiv}{\to}_{\mathsf{e}} \equiv s$ with the same quantitative properties of e''.

3. If u is β_f-normal then it is a fireball (by open harmony, Prop. 2) and so s is $\{\mathsf{m}, \mathsf{e}_\lambda\}$-normal by Lemma 4. By Prop. 4.1, $\to_{\mathsf{e}_y}$ terminates and so there are p and a derivation $g\colon s \to^*_{\mathsf{e}_y} p$ such that p is a e_y-normal form. If p is not a vsub-normal form, then it has a $\{\mathsf{m}, \mathsf{e}_\lambda\}$-redex, but by postponement of $\to_{\mathsf{e}_y}$ (Lemma 5) such a redex was already in s, against hypothesis. So p is a vsub-normal form. Then we have $r \equiv s \to^*_{\mathsf{e}_y} p$. Postponing $\equiv$ (Lemmas 1.2-3), we obtain that there exists a vsub-normal form q and a derivation $f : r \to^*_{\mathsf{e}_y} q \equiv p$.
   To estimate the length of f consider e followed by f, *i.e.* $e; f : t \to^*_{\mathsf{m},\mathsf{e}_\lambda} r \to^*_{\mathsf{e}_y} q$. By Prop. 4.5, $|e; f|_{\mathsf{e}} \le |e; f|_{\mathsf{m}} = |e|_{\mathsf{m}}$, and since $|e; f|_{\mathsf{e}} = |e; f|_{\mathsf{e}_\lambda} + |e; f|_{\mathsf{e}_y} = |e|_{\mathsf{e}_\lambda} + |f|_{\mathsf{e}_y}$ we obtain $|e|_{\mathsf{e}_\lambda} + |f|_{\mathsf{e}_y} \le |e|_{\mathsf{m}}$, *i.e.* $|f|_{\mathsf{e}_y} \le |e|_{\mathsf{m}} - |e|_{\mathsf{e}_\lambda}$. □

Corollary 1 (Linear Termination Equivalence of $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{fire}}$). *Let* $t \in \Lambda$. *There exists a* β_f*-normalizing derivation* d *from* t *iff there exists a* vsub*-normalizing derivation* e *from* t. *Moreover,* $|d|_{\beta_f} \le |e|_{\mathtt{vsub}} \le 2|d|_{\beta_f}$, *i.e. they are linearly related.* See p. 6

Proof.

⇒: Let $d\colon t \to^*_{\beta_f} u$ be a β_f-normalizing derivation and $e\colon t \to^*_{\mathtt{vsub}} \to^*_{\mathsf{e}_y} q$ be the composition of its projection in $\lambda_{\mathtt{vsub}}$ with the extension to a e_y-derivation with q vsub-normal, according to Thm. 1. Then e is a vsub-normalizing derivation from t.

⇐: By contradiction, suppose that there is a diverging β_f-derivation from t in $\lambda_{\mathtt{fire}}$. By Thm. 1 it projects to a vsub-derivation in $\lambda_{\mathtt{vsub}}$ that is at least as long as the one in $\lambda_{\mathtt{fire}}$, absurd.

About lengths, $|d|_{\beta_f} \le |e|_{\mathtt{vsub}}$ since $|e|_{\mathsf{m}} = |d|_{\beta_f}$ (Thm. 1.2). By Prop. 4.5, $|e|_{\mathsf{e}} \le |e|_{\mathsf{m}}$ and so $|e|_{\mathtt{vsub}} = |e|_{\mathsf{m}} + |e|_{\mathsf{e}} \le 2|d|_{\beta_f}$. □

Simulating $\lambda_{\mathtt{sh}}$ ***in*** $\lambda_{\mathtt{vsub}}$

Lemma 22 (Simulation of a sh-Step on $\lambda_{\mathtt{vsub}}$). *Let* $t, u \in \Lambda$.

1. *If* $t \to_{\sigma^\flat} u$ *then there exist* $s, r \in \Lambda_{\mathtt{vsub}}$ *s.t.* $t \to^+_{\mathsf{m}} s \equiv r \overset{+}{\mathsf{m}}\leftarrow u$.
2. *If* $t \to_{\beta^\flat_v} u$ *then there exists* $s \in \Lambda_{\mathtt{vsub}}$ *s.t.* $t \to^+_{\mathsf{m}} \to_{\mathsf{e}} s \overset{*}{\mathsf{m}}\leftarrow u$.

Proof. 1. By induction on the definition of $t \to_{\sigma^\flat} s$, following Remark 3. There are four cases:

(a) *Step at the root*, *i.e.* $t \mapsto_\sigma u$.
   i. either $t := (\lambda x.q)sr \mapsto_{\sigma_1} (\lambda x.qr)s =: u$ with $x \notin \mathtt{fv}(r)$, and then $t = (\lambda x.q)sr \to_{\mathsf{m}} q[x \leftarrow s]r \equiv (qr)[x \leftarrow s] \overset{}{}_{\mathsf{m}}\leftarrow (\lambda x.qr)s = u$;

ii. or $t := v((\lambda x.s)r) \mapsto_{\sigma_3} (\lambda x.vs)r =: u$ with $x \notin \mathtt{fv}(v)$ and then $t = v((\lambda x.s)r) \to_{\mathsf{m}} v(s[x\leftarrow r]) \equiv (vs)[x\leftarrow r]\ {}_{\mathsf{m}}\leftarrow (\lambda x.vs)r = u$.

(b) *Application Left*, *i.e.* $t := sr \to_{\sigma^\flat} qr =: u$ with $s \to_{\sigma^\flat} q$. The result follows by the *i.h.*, as $\to_{\mathsf{m}}$ and $\equiv$ are closed by applicative contexts.

(c) *Application Right*, *i.e.* $t := sr \to_{\sigma^\flat} sq =: u$ with $r \to_{\sigma^\flat} q$. The result follows by the *i.h.*, as $\to_{\mathsf{m}}$ and $\equiv$ are closed by applicative contexts.

(d) *Inside a β-context*, *i.e.* $t := (\lambda x.s)r \to_{\sigma^\flat} (\lambda x.q)r =: u$ with $s \to_{\sigma^\flat} q$. By *i.h.*, $s \to_{\mathsf{m}}^+ s' \equiv q' \ {}_{\mathsf{m}}^+\leftarrow q$. Now, $\to_{\mathsf{m}}$ and $\equiv$ are not closed by balanced contexts, but it is enough to apply a further $\to_{\mathsf{m}}$ step to the balanced context (as $\to_{\mathsf{m}}$ and $\equiv$ are instead closed by substitution contexts), obtaining $t = (\lambda x.s)r \to_{\mathsf{m}} s[x\leftarrow r] \to_{\mathsf{m}}^+ s'[x\leftarrow r] \equiv q'[x\leftarrow r] \ {}_{\mathsf{m}}^+\leftarrow q[x\leftarrow r]\ {}_{\mathsf{m}}\leftarrow (\lambda x.q)r = u$.

2. By induction on the definition of $t \to_{\beta_v^\flat} u$, there are four cases:

(a) *Step at the root*, *i.e.* $t = (\lambda x.r)v \mapsto_{\beta_v} r\{x\leftarrow v\} = u$. So, $t \to_{\mathsf{m}} r[x\leftarrow v] \to_{\mathsf{e}} u$.

(b) *Application Left*. It follows by the *i.h.*, as $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are closed by applicative contexts.

(c) *Application Right*. It follows by the *i.h.*, as $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are closed by applicative contexts.

(d) *Step inside a β-context*, *i.e.* $t = (\lambda x.s)r \to_{\beta_v^\flat} (\lambda x.q)r = u$ with $s \to_{\beta_v^\flat} q$. By *i.h.*, $s \to_{\mathsf{m}}^+ \to_{\mathsf{e}} p \ {}_{\mathsf{m}}^*\leftarrow q$. Now, $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are not closed by balanced contexts, but it is enough to apply a further $\to_{\mathsf{m}}$ step to the balanced context (as $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ are instead closed by substitution contexts), obtaining $(\lambda x.s)r \to_{\mathsf{m}} s[x\leftarrow r] \to_{\mathsf{m}}^+ \to_{\mathsf{e}} p[x\leftarrow r] \ {}_{\mathsf{m}}^*\leftarrow q[x\leftarrow r]\ {}_{\mathsf{m}}\leftarrow (\lambda x.q)r$. □

See p. 6

Lemma 6 (Projecting a sh-Step on $\to_{\mathtt{vsub}_\equiv}$ via m-nf). *Let $t, u \in \Lambda$.*

1. *If $t \to_{\sigma^\flat} u$ then $\mathsf{m}(t) \equiv \mathsf{m}(u)$.*
2. *If $t \to_{\beta_v^\flat} u$ then $\mathsf{m}(t) \to_{\mathsf{e}} \to_{\mathsf{m}}^* \mathsf{m}(u)$.*

Proof.

1. By Lemma 22.1 there exist $s, r \in \Lambda_{\mathtt{vsub}}$ s.t. $t \to_{\mathsf{m}}^+ s \equiv r \ {}_{\mathsf{m}}^+\leftarrow u$. By existence and uniqueness of the m-normal form (Propositions 4.1-2 and Prop. 11.1), $s \to_{\mathsf{m}}^+ \mathsf{m}(s) = \mathsf{m}(t)$. By Lemma 1.2, there is $q \in \Lambda_{\mathtt{vsub}}$ s.t. $r \to_{\mathsf{m}}^+ q \equiv \mathsf{m}(t)$. By Lemma 1.3, q is m-normal; in particular, $q = \mathsf{m}(r) = \mathsf{m}(u)$ according to Prop. 11.1. Thus, $\mathsf{m}(t) \equiv q = \mathsf{m}(u)$.
2. By Lemma 22.2 there are $s, r \in \Lambda_{\mathtt{vsub}}$ such that $t \to_{\mathsf{m}}^+ s \to_{\mathsf{e}} r \ {}_{\mathsf{m}}^*\leftarrow u$. By existence and uniqueness of the m-normal form (Propositions 4.1-2 and Prop. 11.1), $\mathsf{m}(s) = \mathsf{m}(t)$. As $\mathsf{m}(t) \ {}_{\mathsf{m}}^*\leftarrow s \to_{\mathsf{e}} r$, there is $q \in \Lambda_{\mathtt{vsub}}$ s.t. $\mathsf{m}(t) \to_{\mathsf{e}} q \ {}_{\mathsf{m}}^*\leftarrow r$ according to strong commutation of $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$ (Prop. 4.3). Thus, $\mathsf{m}(t) \to_{\mathsf{e}} q \ {}_{\mathsf{m}}^*\leftarrow u$ and hence $\mathsf{m}(t) \to_{\mathsf{e}} \to_{\mathsf{m}}^* \mathsf{m}(u)$ since $\mathsf{m}(u) = \mathsf{m}(q)$ by Prop. 11.1. □

See p. 6

Lemma 7 (Projection Preserves Normal Forms). *Let $t \in \Lambda$. If t is* sh*-normal then $\mathsf{m}(t)$ is* vsub*-normal.*

Proof. In [12, Prop. 12] (where the reduction $\to_{\mathtt{sh}}$ is denoted by $\to_{\mathtt{w}}$) it has been shown that:

1. a term is sh-normal iff it is of the form w,
2. a term is sh-normal and is neither a value nor a β-redex (*i.e.* of the form $(\lambda x.t)u$) iff it is of the form a,

where the forms w and a are defined by mutual induction as follows:

$$a ::= xv \mid xa \mid aw \qquad w ::= v \mid a \mid (\lambda x.w)a.$$

The idea is the following: on the one hand, not only terms of the form a are not values but also they cannot reduce to value through m-derivations; on the other hand, any m-derivation from a term of the form w cannot create an ES of the form $[x\leftarrow L\langle v\rangle]$, therefore the e-normality of w (which is without ES) is preserved in its m-normal form $\mathsf{m}(w)$ and hence $\mathsf{m}(w)$ is vsub-normal.

More formally, consider the types $a_{\mathtt{vsub}}$ and $w_{\mathtt{vsub}}$ of vsub-terms defined by mutual induction as follows (v is a value, without ES):

$$a_{\mathtt{vsub}} ::= xv \mid xa_{\mathtt{vsub}} \mid a_{\mathtt{vsub}}w_{\mathtt{vsub}}$$
$$w_{\mathtt{vsub}} ::= v \mid a_{\mathtt{vsub}} \mid w_{\mathtt{vsub}}[x\leftarrow a_{\mathtt{vsub}}].$$

First, we prove by mutual induction on a and w that the m-normal form $\mathsf{m}(a)$ of a is of the form $a_{\mathtt{vsub}}$, and the m-normal form $\mathsf{m}(w)$ of w is of the form $w_{\mathtt{vsub}}$. The base cases are $\mathsf{m}(v) = v$ (since $\to_{\mathsf{m}}$ does not reduce under λ's) and $\mathsf{m}(xv) = xv$. Inductive cases:

1. $\mathsf{m}(xa) = x\mathsf{m}(a) = xa_{\mathtt{vsub}}$ where $\mathsf{m}(a) = a_{\mathtt{vsub}}$ by *i.h.*,
2. $\mathsf{m}(aw) = \mathsf{m}(a)\mathsf{m}(w) = a_{\mathtt{vsub}}w_{\mathtt{vsub}}$ (since $a_{\mathtt{vsub}}$ is not an abstraction) where $\mathsf{m}(a) = a_{\mathtt{vsub}}$ and $\mathsf{m}(w) = w_{\mathtt{vsub}}$ by *i.h.*,
3. $\mathsf{m}((\lambda x.w)a) = \mathsf{m}(w)[x\leftarrow \mathsf{m}(a)] = w_{\mathtt{vsub}}[x\leftarrow a_{\mathtt{vsub}}]$ (since $a_{\mathtt{vsub}}$ is not of the form $L\langle v\rangle$) where $\mathsf{m}(a) = a_{\mathtt{vsub}}$ and $\mathsf{m}(w) = w_{\mathtt{vsub}}$ by *i.h.*.

To conclude the proof of Lemma 7, it is sufficient to observe that all terms of type $w_{\mathtt{vsub}}$ are vsub-normal, see [5, Lemma 5] (where $\to_{\mathtt{vsub}}$ is denoted by $\to_{\mathtt{w}}$). □

Theorem 2 (Quantitative Simulation of $\lambda_{\mathtt{sh}}$ in $\lambda_{\mathtt{vsub}}$). *Let $t, u \in \Lambda$. If $d\colon t \to_{\mathtt{sh}}^* u$ then there are $s \in \Lambda_{\mathtt{vsub}}$ and $e\colon t \to_{\mathtt{vsub}}^* s$ such that* See p. 6

1. Qualitative Relationship*: $s \equiv \mathsf{m}(u)$;*
2. Quantitative Relationship*: $|d|_{\beta_v^\flat} = |e|_{\mathsf{e}}$;*
3. Normal Forms*: if u is* sh*-normal then s, $\mathsf{m}(u)$ are* vsub*-normal.*

Proof. First, by straightforward induction on $|d|_{\mathtt{sh}} \in \mathbb{N}$ using the projection via m-normal forms (Lemmas 6.1-2), one proves that there is $e_1\colon \mathsf{m}(t) \to_{\mathtt{vsub}_\equiv}^* \mathsf{m}(u)$ with $|e_1|_{\mathsf{e}} = |d|_{\beta_v^\flat}$. By postponement of $\equiv$ (Lemma 1.2), there is $e_2\colon \mathsf{m}(t) \to_{\mathtt{vsub}}^* \equiv \mathsf{m}(u)$ with $|e_2|_{\mathsf{e}} = |e_1|_{\mathsf{e}}$. Clearly, $t \to_{\mathsf{m}}^* \mathsf{m}(t)$. It easy to check that $s \equiv r$ implies $s \not\to_{\mathsf{e}} r$ for all $s, r \in \Lambda_{\mathtt{vsub}}$. Therefore, there exist $s \in \Lambda_{\mathtt{vsub}}$ and $e\colon t \to_{\mathtt{vsub}}^* s$ such that $s \equiv \mathsf{m}(u)$ and $|e|_{\mathsf{e}} = |e_2|_{\mathsf{e}} = |d|_{\beta_v^\flat}$.

Finally, if moreover u is sh-normal then, since normal forms are preserved by multiplicative projection (Lemma 7), $\mathsf{m}(u)$ is vsub-normal, and hence so is s (Lemma 1.3, because $s \equiv \mathsf{m}(u)$). □

Corollary 2 (Termination Equivalence of $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$). *Let $t \in \Lambda$. There is a* sh*-normalizing derivation d from t iff there is a* vsub*-normalizing derivation e from t. Moreover, $|d|_{\beta_v^\flat} = |e|_{\mathsf{e}}$.* See p. 6

Proof.

⇒**:** Let $d\colon t \to_{\mathtt{sh}}^* u$ be a sh-normalizing derivation and $e\colon t \to_{\mathtt{vsub}}^* s$ be its projection in $\lambda_{\mathtt{vsub}}$ with s vsub-normal, according to Thm. 1. Then e is a vsub-normalizing derivation from t.

⇐**:** By contradiction, suppose that there is a diverging sh-derivation d from t in $\lambda_{\mathtt{sh}}$. Since $\to_{\sigma^\flat}$ is strongly normalizing (Prop. 5.2), necessarily in d there are infinitely many $\beta_v^\flat$-steps. By Thm. 1, d projects to a vsub-derivation in $\lambda_{\mathtt{vsub}}$ that has as many e-steps as the $\beta_v^\flat$-steps in $\lambda_{\mathtt{sh}}$, absurd.

About the length, we have $|d|_{\beta_v^\flat} = |e|_{\mathsf{e}}$ by Thm. 1.2. □

Corollary 3 (Number of $\beta_v^\flat$-Steps is Invariant). *All* sh*-normalizing derivations from $t \in \Lambda$ (if any) have the same number of $\beta_v^\flat$-steps.* See p. 6

Proof. Let $d\colon t \to^*_{\mathtt{sh}} u$ and $d'\colon t \to^*_{\mathtt{sh}} u'$ be $\mathtt{sh}$-normalizing. By confluence of $\to_{\mathtt{sh}}$ (Prop. 5.3), $u = u'$. According to Thm. 2, d and d' project, respectively, to two $\mathtt{vsub}$-normalizing derivations $e\colon t \to^*_{\mathtt{vsub}} s \in \Lambda_{\mathtt{vsub}}$ and $e'\colon t \to^*_{\mathtt{vsub}} s' \in \Lambda_{\mathtt{vsub}}$ such that $s \equiv \mathsf{m}(u) \equiv s'$, $|e|_{\mathsf{e}} = |d|_{\beta^\flat_v}$ and $|e'|_{\mathsf{e}} = |d'|_{\beta^\flat_v}$. By Prop. 4.4, $|e|_{\mathsf{e}} = |e'|_{\mathsf{e}}$ and hence $|d|_{\beta^\flat_v} = |d'|_{\beta^\flat_v}$. □

B.3 Proofs of Section 4 (Equational Theories)

See p. 7

Proposition 6. *$\simeq_{\mathtt{vsub}_\equiv}$ is contained in $\simeq_{\beta_f}$ on normalizable terms.*

Proof. Let t and u be normalizable terms (by the results in section Sect. 3 we do not need to specify in which calculus they are normalizing). By confluence of $\to_{\mathtt{vsub}_\equiv}$ (Lemma 1.4) and Prop. 11.1b, $t \simeq_{\mathtt{vsub}_\equiv} u$ implies that there exists s such that $t \to^*_{\mathtt{vsub}_\equiv} s$ and $u \to^*_{\mathtt{vsub}_\equiv} s$, and so t and u have the same $\to_{\mathtt{vsub}_\equiv}$-normal form. By the postponement of $\equiv$ and the fact that it preserves normal forms (Lemma 1) we obtain that t and u have $\equiv$-equivalent $\to_{\mathtt{vsub}}$-normal forms t' and u', respectively.

Now, let t'' and u'' be the normal forms of t and u in $\lambda_{\mathtt{fire}}$, respectively. By Thm. 1, $t'{\downarrow} = t''$ and $u'{\downarrow} = u''$. Since $\equiv$-equivalent terms unfold to the same term, we obtain $t'' = u''$, *i.e.* $t \simeq_{\beta_f} u$. □

Remark 5. For all $\mathtt{vsub}$-term t and $\mathtt{vsub}$-value v, one has $\mathtt{fv}(t) = \mathtt{fv}(t^\circ)$ and $(t\{x{\leftarrow}v\})^\circ = t^\circ\{x{\leftarrow}v^\circ\}$. The proof is by straightforward induction on $t \in \Lambda_{\mathtt{vsub}}$.

See p. 7

Lemma 8. *Projection of $\mathtt{vsub}_\equiv$ on $\lambda_{\mathtt{sh}}$ Let $t, u \in \Lambda_{\mathtt{vsub}}$.*

1. *If $t \to_{\mathsf{m}} u$ then $t^\circ \to^n_{\sigma^\flat_1} u^\circ$ with $n \le |t|_{\mathtt{ES}}$ (so $t^\circ = u^\circ$ if $t \in \Lambda$).*
2. *If $t \to_{\mathsf{e}} u$ then $t^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} u^\circ$ where $n \le |t|_{\mathtt{ES}}$.*
3. *If $t \equiv u$ then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$.*
4. *If $t \to_{\mathtt{vsub}_\equiv} u$ then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$.*

Proof. Both points are proved by induction on $t \in \Lambda_{\mathtt{vsub}}$.

1. According to the definition of $t \to_{\mathsf{m}} u$, there are five cases. The base case is the interesting one, the inductive ones simply follow from the *i.h.*
   - *Step at the root*, *i.e.* $t = L\langle\lambda x.s\rangle r \mapsto_{\mathsf{m}} L\langle s[x{\leftarrow}r]\rangle = u$ where, for some $n \in \mathbb{N}$, $L = \langle\cdot\rangle[x_1{\leftarrow}r_1]\dots[x_n{\leftarrow}r_n]$. We can suppose without loss of generality that $x_i \notin \mathtt{fv}(r)$ for any $1 \le i \le n$. By Remark 5, $x_i \notin \mathtt{fv}(r^\circ)$ for any $1 \le i \le n$. So, $t^\circ = (\lambda x_n.\dots(\lambda x_1 x.s^\circ)r_1^\circ\dots)r_n^\circ r^\circ \to^n_{\sigma^\flat_1} (\lambda x_n.\dots(\lambda x_1.(\lambda x.s^\circ)r^\circ)r_1^\circ\dots)r_n^\circ = u^\circ$ (steps inside a β-context), with $|t|_{\mathtt{ES}} = n+|s|_{\mathtt{ES}}+|r|_{\mathtt{ES}}+\sum_{i=1}^n |r_i|_{\mathtt{ES}} \ge n$.
   - *Application Left*, *i.e.* $t = sr \to_{\mathsf{m}} qr = u$ with $s \to_{\mathsf{m}} q$. By *i.h.* $s^\circ \to^n_{\sigma^\flat_1} q^\circ$ for some $n \le |s|_{\mathtt{ES}}$, therefore $t^\circ = s^\circ r^\circ \to^n_{\sigma^\flat_1} q^\circ r^\circ = u^\circ$ where $|t|_{\mathtt{ES}} = |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
   - *Application Right*, *i.e.* $t = sr \to_{\mathsf{m}} sq = u$ with $r \to_{\mathsf{m}} q$. By *i.h.* $r^\circ \to^n_{\sigma^\flat_1} q^\circ$ for some $n \le |r|_{\mathtt{ES}}$, hence $t^\circ = s^\circ r^\circ \to^n_{\sigma^\flat_1} s^\circ q^\circ = u^\circ$ where $|t|_{\mathtt{ES}} = |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
   - *ES Left*, *i.e.* $t = s[x{\leftarrow}r] \to_{\mathsf{m}} q[x{\leftarrow}r] = u$ with $s \to_{\mathsf{m}} q$. By *i.h.* $s^\circ \to^n_{\sigma^\flat_1} q^\circ$ for some $n \le |s|_{\mathtt{ES}}$, thus $t^\circ = (\lambda x.s^\circ)r^\circ \to^n_{\sigma^\flat_1} (\lambda x.q^\circ)r^\circ = u^\circ$ (steps inside a β-context) where $|t|_{\mathtt{ES}} = 1 + |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
   - *ES Right*, *i.e.* $t = s[x{\leftarrow}r] \to_{\mathsf{m}} s[x{\leftarrow}q] = u$ with $r \to_{\mathsf{m}} q$. By *i.h.* $r^\circ \to^n_{\sigma^\flat_1} q^\circ$ for some $n \le |r|_{\mathtt{ES}}$, therefore $t^\circ = (\lambda x.s^\circ)r^\circ \to^n_{\sigma^\flat_1} (\lambda x.s^\circ)q^\circ = u^\circ$ where $|t|_{\mathtt{ES}} = 1 + |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
2. According to the definition of $t \to_{\mathsf{e}} u$, there are five cases. The base case is the interesting one, the inductive ones simply follow from the *i.h.*
   - *Step at the root*, *i.e.* $t = s[x{\leftarrow}L\langle v\rangle] \mapsto_{\mathsf{e}} L\langle s\{x{\leftarrow}v\}\rangle = u$ where, for some $n \in \mathbb{N}$, $L = \langle\cdot\rangle[x_1{\leftarrow}r_1]\dots[x_n{\leftarrow}r_n]$. We can suppose without loss of generality that $x_i \notin \mathtt{fv}(s)$ for any $1 \le i \le n$, and $(s\{x{\leftarrow}v\})^\circ = s^\circ\{x{\leftarrow}v^\circ\}$. Therefore,
     $$\begin{aligned} t^\circ &= (\lambda x.s^\circ)(\lambda x_n.\dots(\lambda x_1.v^\circ)r_1^\circ\dots)r_n^\circ \\ &\to^n_{\sigma^\flat_3} (\lambda x_n.\dots(\lambda x_1.(\lambda x.s^\circ)v^\circ)r_1^\circ\dots)r_n^\circ \\ &\to_{\beta^\flat_v} (\lambda x_n.\dots(\lambda x_1.s^\circ\{x{\leftarrow}v^\circ\})r_1^\circ\dots)r_n^\circ = u^\circ \end{aligned}$$
     (steps inside a β-context) where $|t|_{\mathtt{ES}} = n + |s|_{\mathtt{ES}} + |v|_{\mathtt{ES}} + \sum_{i=1}^n |r_i|_{\mathtt{ES}} \ge n$.
   - *Application Left*, *i.e.* $t = sr \to_{\mathsf{e}} qr = u$ with $s \to_{\mathsf{e}} q$. By *i.h.* $s^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} q^\circ$ for some $n \le |s|_{\mathtt{ES}}$, thus $t^\circ = s^\circ r^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} q^\circ r^\circ = u^\circ$ with $|t|_{\mathtt{ES}} = |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
   - *Application Right*, *i.e.* $t = sr \to_{\mathsf{e}} sq = u$ with $r \to_{\mathsf{e}} q$. By *i.h.* $r^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} q^\circ$ for some $n \le |r|_{\mathtt{ES}}$, so $t^\circ = s^\circ r^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} s^\circ q^\circ = u^\circ$ with $|t|_{\mathtt{ES}} = |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
   - *ES Left*, *i.e.* $t = s[x{\leftarrow}r] \to_{\mathsf{e}} q[x{\leftarrow}r] = u$ with $s \to_{\mathsf{e}} q$. By *i.h.* $s^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} q^\circ$ where $n \le |s|_{\mathtt{ES}}$, thus $t^\circ = (\lambda x.s^\circ)r^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} (\lambda x.q^\circ)r^\circ = u^\circ$ (steps inside a β-context) where $|t|_{\mathtt{ES}} = 1 + |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
   - *ES Right*, *i.e.* $t = s[x{\leftarrow}r] \to_{\mathsf{e}} s[x{\leftarrow}q] = u$ with $r \to_{\mathsf{e}} q$. By *i.h.* $r^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} q^\circ$ for some $n \le |r|_{\mathtt{ES}}$, therefore $t^\circ = (\lambda x.s^\circ)r^\circ \to^n_{\sigma^\flat_3}\to_{\beta^\flat_v} (\lambda x.s^\circ)q^\circ = u^\circ$ where $|t|_{\mathtt{ES}} = 1 + |s|_{\mathtt{ES}} + |r|_{\mathtt{ES}} \ge n$.
3. First, let $\equiv'$ be the symmetric closure under evaluation contexts (see Fig. 3) of the binary relation $\equiv_{\mathtt{com}} \cup \equiv_{@\mathsf{l}} \cup \equiv_{@\mathsf{r}} \cup \equiv_{[\cdot]}$ on $\Lambda_{\mathtt{vsub}}$. We prove by induction on $t \in \Lambda_{\mathtt{vsub}}$ the following fact:
   $$\text{if } t \equiv' u \text{ then } t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ \tag{4}$$
   According to the definition of $\equiv'$, there are five cases:
   - *Axioms*, *i.e.* $t \equiv_{\mathsf{r}} u$ or $u \equiv_{\mathsf{r}} t$ for some $\mathsf{r} \in \{\mathtt{com}, @_{\mathsf{l}}, @_{\mathsf{r}}, [\cdot]\}$. As $\simeq^{\mathsf{ext}}_{\mathtt{sh}}$ is symmetric, it is enough to suppose $t \equiv_{\mathsf{r}} u$. Cases:
     - if $t := s[x{\leftarrow}r][y{\leftarrow}q] \equiv_{\mathtt{com}} s[y{\leftarrow}q][x{\leftarrow}r] =: u$ with $x \notin \mathtt{fv}(q)$ and $y \notin \mathtt{fv}(r)$, then $t^\circ = (\lambda y.(\lambda x.s^\circ)r^\circ)q^\circ \mapsto_{\sigma_{\mathtt{com}}} (\lambda x.(\lambda y.s^\circ)q^\circ)r^\circ = u^\circ$ since $x \notin \mathtt{fv}(q^\circ)$ and $y \notin \mathtt{fv}(r^\circ)$ according to Remark 5, therefore $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$;
     - if $t := qs[x{\leftarrow}r] \equiv_{@\mathsf{r}} (qs)[x{\leftarrow}r] =: u$ with $x \notin \mathtt{fv}(q)$, then $t^\circ = q^\circ((\lambda x.s^\circ)r^\circ) \mapsto_{\sigma'_3} (\lambda x.q^\circ s^\circ)r^\circ = u^\circ$ since $x \notin \mathtt{fv}(q^\circ)$ by Remark 5, thus $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$;
     - if $t := q[x{\leftarrow}r]s \equiv_{@\mathsf{l}} (qs)[x{\leftarrow}r] =: u$ with $x \notin \mathtt{fv}(s)$, then $t^\circ = (\lambda x.q^\circ)r^\circ s^\circ \mapsto_{\sigma_1} (\lambda x.q^\circ s^\circ)r^\circ = u^\circ$ since $x \notin \mathtt{fv}(s^\circ)$ by Remark 5, so $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$;
     - if $t := s[x{\leftarrow}r[y{\leftarrow}q]] \equiv_{[\cdot]} s[x{\leftarrow}r][y{\leftarrow}q] =: u$ with $y \notin \mathtt{fv}(s)$, then $t^\circ = (\lambda x.s^\circ)((\lambda y.r^\circ)q^\circ) \mapsto_{\sigma_3} (\lambda y.(\lambda x.s^\circ)r^\circ)q^\circ = u^\circ$ since $y \notin \mathtt{fv}(s^\circ)$ according to Remark 5, so $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$.
   - *Application Left*, *i.e.* $t := sr \equiv' qr =: u$ with $s \equiv' q$: by *i.h.*, $s^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} q^\circ$ and hence $t^\circ = s^\circ r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} q^\circ r^\circ = u^\circ$.
   - *Application Right*, *i.e.* $t := sr \equiv' sq =: u$ with $r \equiv' q$: by *i.h.*, $r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} q^\circ$ and hence $t^\circ = s^\circ r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} s^\circ q^\circ = u^\circ$.
   - *ES Left*, *i.e.* $t := s[x{\leftarrow}r] \equiv' q[x{\leftarrow}r] =: u$ with $s \equiv' q$: by *i.h.*, $s^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} q^\circ$ and thus $t^\circ = (\lambda x.s^\circ)r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} (\lambda x.q^\circ)r^\circ = u^\circ$.
   - *ES Right*, *i.e.* $t := s[x{\leftarrow}r] \equiv' s[x{\leftarrow}q] =: u$ with $r \equiv' q$: by *i.h.*, $r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} q^\circ$ and hence $t^\circ = (\lambda x.s^\circ)r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} (\lambda x.s^\circ)q^\circ = u^\circ$.

   To conclude the proof of Lemma 8.3 it is sufficient to observe that $\equiv$ is the reflexive-transitive closure of $\equiv'$, and to show, by straightforward induction on $n \in \mathbb{N}$ and using fact (4), that if $t \equiv'^n u$ then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$.

4. By definition, $t \to_{\mathtt{vsub}_\equiv} u$ means that: See p. 8
   - either $t \equiv s \to_{\mathsf{m}} r \equiv u$ for some $s, r \in \Lambda_{\mathtt{vsub}}$, and then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} s^\circ \to^*_{\sigma_1^\flat} r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$ by Lemmas 8.1 and 8.3,
   - or $t \equiv s \to_{\mathsf{e}} r \equiv u$ for some $s, r \in \Lambda_{\mathtt{vsub}}$, and then $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} s^\circ \to^*_{\sigma_3^\flat} \to_{\beta_v^\flat} r^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$ by Lemmas 8.2-3.

   In all cases we have $t^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} u^\circ$. □

See p. 7 **Theorem 3** (Same Equational Theory for $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$). *Let $t, u \in \Lambda$: $t \simeq_{\mathtt{vsub}_\equiv} u$ iff $t \simeq^{\mathsf{ext}}_{\mathtt{sh}} u$.*

Proof. ⇒: By definition, $t \simeq_{\mathtt{vsub}_\equiv} u$ means that, for some $n \in \mathbb{N}$, there are $s_1, \dots, s_n \in \Lambda_{\mathtt{vsub}}$ such that $s_0 = t$, $s_n = u$ and $sj \to_{\mathtt{vsub}_\equiv} s_{j+1}$ or $s_{j+1} \to_{\mathtt{vsub}_\equiv} s_j$ for any $0 \le j < n$. By Lemma 8.4, $s_j^\circ \simeq^{\mathsf{ext}}_{\mathtt{sh}} s_{j+1}^\circ$ for any $0 \le j < n$. Since $t, u \in \Lambda$ (*i.e.* they are without ES), $t^\circ = t$ and $u^\circ = u$. Therefore, $t \simeq^{\mathsf{ext}}_{\mathtt{sh}} u$.

⇐: First, let $\sim_{\mathtt{sh}}$ be the symmetric closure under balanced contexts (see Fig. 4) of the binary relation $\mapsto_{\sigma_{\mathtt{com}}} \cup \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3'} \cup \mapsto_{\beta_v}$ on Λ. We prove by induction on $t \in \Lambda$ the following fact:

$$\text{if } t \sim_{\mathtt{sh}} u \text{ then } t \simeq_{\mathtt{vsub}_\equiv} u \tag{5}$$

According to the definition of $\sim_{\mathtt{sh}}$, there are four cases:

- *Axioms*, *i.e.* $t \mapsto_{\sigma_{\mathtt{com}}} u$ or $t \mapsto_{\sigma_1} u$ or $t \mapsto_{\sigma_3'} u$ or $t \mapsto_{\beta_v} u$ or $u \mapsto_{\sigma_{\mathtt{com}}} t$ or $u \mapsto_{\sigma_1} t$ or $u \mapsto_{\sigma_3'} t$ or $u \mapsto_{\beta_v} t$. As $\simeq_{\mathtt{vsub}_\equiv}$ is symmetric, it is enough to consider the following cases:
  - if $t := (\lambda y.(\lambda x.s)r)q \mapsto_{\sigma_{\mathtt{com}}} (\lambda x.(\lambda y.s)q)r =: u$ with $x \notin \mathtt{fv}(q)$ and $y \notin \mathtt{fv}(r)$, then $t \to^+_{\mathsf{m}} s[x{\leftarrow}r][y{\leftarrow}q] \equiv_{\mathtt{com}} s[y{\leftarrow}q][x{\leftarrow}r] \ {}_{\mathsf{m}}{\leftarrow} u$, therefore $t \simeq_{\mathtt{vsub}_\equiv} u$;
  - if $t := q((\lambda x.s)r) \mapsto_{\sigma_3'} (\lambda x.qs)r =: u$ with $x \notin \mathtt{fv}(q)$, then $t \to_{\mathsf{m}} q\, s[x{\leftarrow}r] \equiv_{@\mathsf{r}} (qs)[x{\leftarrow}r] \ {}_{\mathsf{m}}{\leftarrow} u$, so $t \simeq_{\mathtt{vsub}_\equiv} u$;
  - if $t := (\lambda x.q)rs \mapsto_{\sigma_1} (\lambda x.qs)r =: u$ with $x \notin \mathtt{fv}(s)$, then $t \to_{\mathsf{m}} q[x{\leftarrow}r]s \equiv_{@\mathsf{l}} (qs)[x{\leftarrow}r] \ {}_{\mathsf{m}}{\leftarrow} u$, thus $t \simeq_{\mathtt{vsub}_\equiv} u$;
  - if $t := (\lambda x.s)v \mapsto_{\beta_v} s\{x{\leftarrow}v\} =: u$, then $t \to_{\mathsf{m}} s[x{\leftarrow}v] \to_{\mathsf{e}} u$ and hence $t \simeq_{\mathtt{vsub}_\equiv} u$.
- *Application Left*, *i.e.* $t := sr \sim_{\mathtt{sh}} qr =: u$ with $s \sim_{\mathtt{sh}} q$: by *i.h.*, $s \simeq_{\mathtt{vsub}_\equiv} q$ and hence $t = sr \simeq_{\mathtt{vsub}_\equiv} qr = u$.
- *Application Right*, *i.e.* $t := sr \sim_{\mathtt{sh}} sq =: u$ with $r \sim_{\mathtt{sh}} q$: by *i.h.*, $r \simeq_{\mathtt{vsub}_\equiv} q$ and hence $t = sr \simeq_{\mathtt{vsub}_\equiv} sq = u$.
- *Inside a β-context*, *i.e.* $t := (\lambda x.s)r \sim_{\mathtt{sh}} (\lambda x.q)r =: u$ with $s \sim_{\mathtt{sh}} q$: by *i.h.*, $s \simeq_{\mathtt{vsub}_\equiv} q$ and thus $t \to_{\mathsf{m}} s[x{\leftarrow}r] \simeq_{\mathtt{vsub}_\equiv} q[x{\leftarrow}r] \ {}_{\mathsf{m}}{\leftarrow} u$, therefore $t \simeq_{\mathtt{vsub}_\equiv} u$.

To conclude the proof of Thm. 3 it is sufficient to observe that $\simeq^{\mathsf{ext}}_{\mathtt{sh}}$ is the reflexive-transitive closure of $\sim_{\mathtt{sh}}$, and to show, by straightforward induction on $n \in \mathbb{N}$ and using fact (5), that if $t \sim^n_{\mathtt{sh}} u$ then $t \simeq_{\mathtt{vsub}_\equiv} u$. □

B.4 Proofs of Section 5 (How to Stop Worrying and Love the Bomb)

See p. 7 **Proposition 7** (Abstraction Size-Explosion). *For any $n > 0$, one has $t_n v \to^n_{\beta_\lambda} R^v_n$ with $|t_n| = O(n)$, $|R^v_n| = O(2^n)$, and R^v_n is β_f-normal.*

Proof. By induction on $n \in \mathbb{N}$, we prove more generally that $t_n R^v_m \to^n_{\beta_\lambda} R^v_{m+n}$ for all $n, m \in \mathbb{N}$. Base case: $t_1 R^v_m \to_\lambda \lambda y.(y R^v_m R^v_m) = R^v_{m+1}$. Inductive case:

$$t_{n+1} R^v_m \to_{\beta_\lambda} t_n\ (\lambda y.(y R^v_m R^v_m)) = t_n R^v_{m+1} \overset{i.h.}{\to}{}^n_{\beta_\lambda} R^v_{m+n+1}$$

The part about sizes and R^v_n being a normal form is immediate. □

Proposition 8 (Inert Size-Explosion). *For any $n > 0$, $u_n i \to^n_{\beta_i} i^{2^n}$ (the β_f-normal form of $u_n i$) with $|u_n| = O(n)$ and $|i^{2^n}| = O(2^n)$.*

Proof. By induction on $n \in \mathbb{N}$. Let $i' := i^2 = ii$. Cases:

$$\begin{array}{lcll} u_1 & = & (\lambda x_1.(x_1 x_1))i & \to_{\beta_i} \ i^2 \\ u_{n+1} & = & (\lambda x_{n+1}.(u_n(x_{n+1}x_{n+1})))i & \to_{\beta_i} \\ & & u_n i^2 = u_n i' & \to^n_{\beta_i} \ (\textit{i.h.}) \\ & & i'^{2^n} & = \ i^{2^{n+1}} \end{array}$$

□

B.5 Proofs of Section 6 (Easy GLAMOUr)

Lemma 9 (Properties of $\to_{\mathtt{r}\beta_f}$). *Let $t \in \Lambda$.* See p. 8

1. Completeness: *t has $\to_{\beta_f}$-redex iff t has a $\to_{\mathtt{r}\beta_f}$-redex.*
2. Determinism: *t has at most one $\to_{\mathtt{r}\beta_f}$ redex.*

Proof.

1. ⇒) Immediate, as right contexts are in particular evaluation contexts, and so $\to_{\mathtt{r}\beta_f} \subseteq \to_{\beta_f}$. ⇐) Let E the evaluation context of the rightmost redex of t. We show that E is a right context. By induction on E. Cases:
   (a) *Empty*, *i.e.* $E = \langle\cdot\rangle$. Then clearly E is a right context.
   (b) *Right Application*, *i.e.* $t = us$ and $E = uE'$. By *i.h.* E' is a right context in s and so is E with respect to t.
   (c) *Left Application*, *i.e.* $t = us$ and $E = E's$. By *i.h.* E' is a right context in u. Since E is the rightmost evaluation context, s is $\to_{\beta_f}$-normal, and so by open harmony (Prop. 2) it is a fireball. Therefore E is a right context.
2. By induction on t. Note that by completeness of $\to_{\mathtt{r}\beta_f}$ (Point 1) open harmony (Prop. 2) holds with respect to $\to_{\mathtt{r}\beta_f}$, *i.e.* a term is $\to_{\mathtt{r}\beta_f}$-normal iff it is a fireball. We use this fact implicitly in the following case analysis. Cases:
   - *Value*. No redexes.
   - *Application $t = us$*. By *i.h.*, there are two cases for s:
     (a) *s has exactly one $\to_{\mathtt{r}\beta_f}$ redex*. Then t has a $\to_{\mathtt{r}\beta_f}$ redex, because $u\langle\cdot\rangle$ is an evaluation context. Moreover, no $\to_{\mathtt{r}\beta_f}$ redex for t can lie in u, because by open harmony s is not a fireball, and so $\langle\cdot\rangle s$ is not a right context.
     (b) *s has no $\to_{\mathtt{r}\beta_f}$ redexes*. Then s is a fireball. Consider u. By *i.h.*, there are two cases:
       i. *u has exactly one $\to_{\mathtt{r}\beta_f}$ redex*. Then t has a $\to_{\mathtt{r}\beta_f}$ redex, because $\langle\cdot\rangle s$ is an evaluation context and s is a fireball. Uniqueness follows from the fact that s has no $\to_{\mathtt{r}\beta_f}$ redexes.
       ii. *u has no $\to_{\mathtt{r}\beta_f}$ redexes*. By open harmony u is a fireball, and there are two cases:
         - *u is a inert term i or a variable x*. Then t is a fireball.
         - *u is an abstraction $\lambda x.r$*. Then $t = (\lambda x.r)s$ is a $\to_{\mathtt{r}\beta_f}$-redex, because s is a fireball. Moreover, there are no other $\to_{\mathtt{r}\beta_f}$ redexes, because evaluation does not go under abstractions and s is a fireball. □

Lemma 10 (Easy GLAMOUr Invariants). *Let $s = (D, \overline{t}, \pi, E)$ be a reachable state. Then:* See p. 9

1. Name:
   (a) Substitutions: *if $E = E' : [x{\leftarrow}\overline{u}] : E''$ then x is fresh wrt $\overline{u}$ and E'';*
   (b) Abstractions: *if $\lambda x.\overline{s}$ is a subterm of D, $\overline{u}$, π, or E then x may occur only in $\overline{s}$;*
2. Fireball Item: *$\underline{\phi}\sigma_E$ is a inert term if $\phi = x@\pi'$ and an abstraction otherwise, for every item ϕ in π, in E, and in every stack in D;*

3. Contextual Decoding: $E_s = \underline{D}\langle\underline{\pi}\rangle\sigma_E$ *is a right context;*

Proof. By induction on the length of the execution leading to the reachable state. In an initial state all the invariants trivially hold. For a non-empty execution the proof for every invariant is by case analysis on the last transition, using the *i.h.*.

1. *Name*. Cases:
   (a) $s' = (D, \bar{t}\overline{u}, \pi, E) \rightsquigarrow_{\mathtt{c}_1} (D : \bar{t}\Diamond\pi, \overline{u}, \epsilon, E) = s$. Both points follow immediately from the *i.h.*
   (b) $s' = (D : \bar{t}\Diamond\pi, \lambda x.\overline{u}, \epsilon, E) \rightsquigarrow_{\mathtt{c}_2} (D, \bar{t}, \lambda x.\overline{u}@\epsilon : \pi, E) = s$. Both points follow immediately from the *i.h.*
   (c) $s' = (D : \bar{t}\Diamond\pi, x, \pi', E_1[x\leftarrow y@\pi'']E_2) \rightsquigarrow_{\mathtt{c}_3} (D, \bar{t}, x@\pi' : \pi, E_1[x\leftarrow y@\pi'']E_2) = s$. Both points follow immediately from the *i.h.*
   (d) $s' = (D : \bar{t}\Diamond\pi, x, \pi', E) \rightsquigarrow_{\mathtt{c}_4} (D, \bar{t}, x@\pi' : \pi, E) = s$ with $E(x) = \bot$. Both points follow immediately from the *i.h.*
   (e) $s' = (D, \lambda x.\bar{t}, \phi : \pi, E) \rightsquigarrow_{\mathtt{m}} (D, \bar{t}, \pi, [x\leftarrow\phi]E) = s$. Point 1 for the new entry in the environment follows from the *i.h.* for Point 2, for the other entries from the *i.h.* for Point 1. Point 2 follows from its *i.h.*
   (f)
   $$s' = (D, x, \pi, E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2) \rightsquigarrow_{\mathtt{e}} (D, (\lambda y.\overline{u})^{\alpha}, \pi, E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2) = s.$$
   Point 1 follows from its *i.h.*. Point 2 for the new code is guaranteed by the α-renaming operation $(\lambda y.\overline{u})^{\alpha}$, the rest follows from its *i.h.*
2. *Fireball Item*. Cases:
   (a) $s' = (D, \bar{t}\overline{u}, \pi, E) \rightsquigarrow_{\mathtt{c}_1} (D : \bar{t}\Diamond\pi, \overline{u}, \epsilon, E) = s$. It follows from the *i.h.*
   (b) $s' = (D : \bar{t}\Diamond\pi, \lambda x.\overline{u}, \epsilon, E) \rightsquigarrow_{\mathtt{c}_2} (D, \bar{t}, \lambda x.\overline{u}@\epsilon : \pi, E) = s$. For $\lambda x.\overline{u}@\epsilon$ we have that $\underline{\lambda x.\overline{u}@\epsilon}\sigma_E = (\lambda x.\overline{u})\sigma_E = \lambda x.\overline{u}\sigma_E$ is an abstraction, *i.e.* a fireball. For all other items the invariant follows from the *i.h.*
   (c) $s' = (D : \bar{t}\Diamond\pi, x, \pi', E_1[x\leftarrow y@\pi'']E_2) \rightsquigarrow_{\mathtt{c}_3} (D, \bar{t}, x@\pi' : \pi, E_1[x\leftarrow y@\pi'']E_2) = s$. Let $E' := E_1[x\leftarrow y@\pi'']E_2$. For $x@\pi'$ we have that $\underline{x@\pi'}\sigma_{E'} = \langle x\sigma_{E'}\rangle(\underline{\pi'}\sigma_{E'})$. Now, by Point 1 it follows that every ES in E' binds a different variable, and so $x\sigma_{E'} = x\sigma_{E_1[x\leftarrow y@\pi'']E_2} = x\sigma_{E_1}\{x\leftarrow\underline{y@\pi''}\}\sigma_{E_2} = \underline{y@\pi''}\sigma_{E_2}$, that by *i.h.* is a inert term. Moreover, the *i.h.* also gives that $\phi'\sigma_{E'}$ is a fireball for every item ϕ' in π'. Therefore $\underline{x@\pi'}\sigma_{E'} = \langle x\sigma_{E'}\rangle(\underline{\pi'}\sigma_{E'})$ is a inert term. For all other items in s the invariant follows from the *i.h.*
   (d) $s' = (D : \bar{t}\Diamond\pi, x, \pi', E) \rightsquigarrow_{\mathtt{c}_4} (D, \bar{t}, x@\pi' : \pi, E) = s$ with $E(x) = \bot$. Similar to the previous case. For $x@\pi'$ we have that $\underline{x@\pi'}\sigma_E = \langle x\sigma_E\rangle(\underline{\pi'}\sigma_E)$. Now, since by hypothesis $E(x) = \bot$, we have $x\sigma_E = x$. As before, by *i.h.* $\phi'\sigma_E$ is a fireball for every item ϕ' in π'. Therefore $\underline{x@\pi'}\sigma_{E'}$ is a inert term. For all other items in s the invariant follows from the *i.h.*
   (e) $s' = (D, \lambda x.\bar{t}, \phi : \pi, E) \rightsquigarrow_{\mathtt{m}} (D, \bar{t}, \pi, [x\leftarrow\phi]E) = s$. By Point 2 x may occur only in $\bar{t}$. Thus the substitution $\sigma_{[x\leftarrow\phi]E}$ acts exactly as σ_E on every item in s. Then the invariant follows from the *i.h.*.
   (f)
   $$s' = (D, x, \pi, E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2) \rightsquigarrow_{\mathtt{e}} (D, (\lambda y.\overline{u})^{\alpha}, \pi, E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2) = s.$$
   It follows from the *i.h.*
3. *Contextual Decoding*. Cases:
   (a) $s' = (D, \bar{t}\overline{u}, \pi, E) \rightsquigarrow_{\mathtt{c}_1} (D : \bar{t}\Diamond\pi, \overline{u}, \epsilon, E) = s$. By *i.h.* $E_{s'} = \underline{D}\langle\underline{\pi}\rangle\sigma_E$ is a right context, as well as $\overline{u}\sigma_E\langle\cdot\rangle$. Then their composition $(\underline{D}\langle\underline{\pi}\rangle\sigma_E)\langle\overline{u}\sigma_E\langle\cdot\rangle\rangle = \underline{D}\langle\langle\overline{u}\langle\cdot\rangle\rangle\underline{\pi}\rangle\sigma_E = E_s$ is a right context.
   (b) $s' = (D : \bar{t}\Diamond\pi, \lambda x.\overline{u}, \epsilon, E) \rightsquigarrow_{\mathtt{c}_2} (D, \bar{t}, \lambda x.\overline{u}@\epsilon : \pi, E) = s$. By *i.h.* $E_{s'} = \underline{D : \bar{t}\Diamond\pi}\sigma_E = \underline{D}\langle\langle\bar{t}\langle\cdot\rangle\rangle\underline{\pi}\rangle\sigma_E$ is a right context, that implies that $\underline{D}\langle\underline{\pi}\rangle\sigma_E$ is one such context as well. Then $E_{s'} = \underline{D}\langle\underline{\lambda x.\overline{u}@\epsilon : \pi}\rangle\sigma_E = \underline{D}\langle\langle\langle\cdot\rangle\lambda x.\overline{u}\rangle\underline{\pi}\rangle\sigma_E = (\underline{D}\langle\underline{\pi}\rangle\sigma_E)\langle\langle\cdot\rangle\lambda x.\overline{u}\sigma_E\rangle$ is a right context , because it is the composition of right context, given that $\lambda x.\overline{u}\sigma_E$ is a fireball.
   (c) $s' = (D : \bar{t}\Diamond\pi, x, \pi', E_1[x\leftarrow y@\pi'']E_2) \rightsquigarrow_{\mathtt{c}_3} (D, \bar{t}, x@\pi' : \pi, E_1[x\leftarrow y@\pi'']E_2) = s$. Let $E := E_1[x\leftarrow y@\pi'']E_2$. By *i.h.* $E_{s'} = \underline{D : \bar{t}\Diamond\pi\langle\underline{\pi'}\rangle}\sigma_E = \underline{D}\langle\langle\bar{t}(\underline{\pi'})\rangle\underline{\pi}\rangle\sigma_E$ is a right context, that implies that $\underline{D}\langle\underline{\pi}\rangle\sigma_E$ is one such context as well. Then $E_s = \underline{D}\langle\underline{x@\pi' : \pi}\rangle\sigma_E = \underline{D}\langle\langle\langle\cdot\rangle\underline{x@\pi'}\rangle\underline{\pi}\rangle\sigma_E = (\underline{D}\langle\underline{\pi}\rangle\sigma_E)\langle\langle\cdot\rangle\underline{x@\pi'}\sigma_E\rangle$ is a right context, because it is the composition of right context, given that $\underline{x@\pi'}\sigma_E$ is a fireball by Point 3.
   (d) $s' = (D : \bar{t}\Diamond\pi, x, \pi', E) \rightsquigarrow_{\mathtt{c}_4} (D, \bar{t}, x@\pi' : \pi, E) = s$ with $E(x) = \bot$. Exactly as the previous case.
   (e) $s' = (D, \lambda x.\bar{t}, \phi : \pi, E) \rightsquigarrow_{\mathtt{m}} (D, \bar{t}, \pi, [x\leftarrow\phi]E) = s$. By *i.h.* $E_{s'} = \underline{D}\langle\underline{\phi : \pi}\rangle\sigma_E$ is a right context, that implies that $\underline{D}\langle\underline{\pi}\rangle\sigma_E$ is one such context as well. Now, note that $E_{s'} = \underline{D}\langle\underline{\pi}\rangle\sigma_{[x\leftarrow\phi]E} = \underline{D}\langle\underline{\pi}\rangle\sigma_E$ because by Point 2 x may occur only in $\bar{t}$, and so the substitution $\sigma_{[x\leftarrow\phi]E}$ acts on every code in D and π exactly as σ_E.
   (f)
   $$s' = (D, x, \pi, E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2) \rightsquigarrow_{\mathtt{e}} (D, (\lambda y.\overline{u})^{\alpha}, \pi, E_1[x\leftarrow\lambda y.\overline{u}@\epsilon]E_2) = s.$$
   It follows by the *i.h.* because $E_{s'} = E_s$, as the only component that changes is the code. □

Lemma 11 (Easy GLAMOUr One-Step Weak Simulation)**.** *Let* s See p. 9 *be a reachable state.*

1. Commutative & Exponential*: if* $s \rightsquigarrow_{\mathtt{e},\mathtt{c}_{1,2,3,4}} s'$ *then* $\underline{s} = \underline{s'}$*;*
2. Multiplicative*: if* $s \rightsquigarrow_{\mathtt{m}} s'$ *then* $\underline{s} \rightarrow_{\mathtt{r}\beta_f} \underline{s'}$*.*

Proof. Transitions:

1. $s = (D, \bar{t}\overline{u}, \pi, E) \rightsquigarrow_{\mathtt{c}_1} (D : \bar{t}\Diamond\pi, \overline{u}, \epsilon, E) = s'$. Then
   $$\begin{array}{rcl}\underline{s} & = & \underline{D}\langle\langle\bar{t}\overline{u}\rangle\underline{\pi}\rangle\sigma_E \\ & = & \underline{D : \bar{t}\Diamond\pi}\langle\overline{u}\rangle\sigma_E \\ & = & \underline{D : \bar{t}\Diamond\pi}\langle\langle\overline{u}\rangle\underline{\epsilon}\rangle\sigma_E \quad = \quad \underline{s'}\end{array}$$
2. $s = (D : \bar{t}\Diamond\pi, \lambda x.\overline{u}, \epsilon, E) \rightsquigarrow_{\mathtt{c}_2} (D, \bar{t}, \lambda x.\overline{u}@\epsilon : \pi, E) = s'$. Then
   $$\begin{array}{rcl}\underline{s} & = & \underline{D : \bar{t}\Diamond\pi}\langle\langle\lambda x.\overline{u}\rangle\underline{\epsilon}\rangle\sigma_E \\ & = & \underline{D}\langle\langle\bar{t}(\langle\lambda x.\overline{u}\rangle\underline{\epsilon})\rangle\underline{\pi}\rangle\sigma_E \\ & = & \underline{D}\langle\langle\bar{t}\rangle\underline{\lambda x.\overline{u}@\epsilon : \pi}\rangle\sigma_E \quad = \quad \underline{s'}\end{array}$$
3. $s = (D : \bar{t}\Diamond\pi, x, \pi', E_1[x\leftarrow y@\pi'']E_2) \rightsquigarrow_{\mathtt{c}_3} (D, \bar{t}, x@\pi' : \pi, E_1[x\leftarrow y@\pi'']E_2) = s'$. Let $E' := E_1[x\leftarrow y@\pi'']E_2$ Then
   $$\begin{array}{rcl}\underline{s} & = & \underline{D : \bar{t}\Diamond\pi}\langle\langle x\rangle\underline{\pi'}\rangle\sigma_{E'} \\ & = & \underline{D}\langle\langle\bar{t}(\langle x\rangle\underline{\pi'})\rangle\underline{\pi}\rangle\sigma_{E'} \\ & = & \underline{D}\langle\langle\bar{t}\rangle\underline{x@\pi' : \pi}\rangle\sigma_{E'} \quad = \quad \underline{s'}\end{array}$$
4. $s = (D : \bar{t}\Diamond\pi, x, \pi', E) \rightsquigarrow_{\mathtt{c}_4} (D, \bar{t}, x@\pi' : \pi, E) = s'$ with $E(x) = \bot$. Then
   $$\begin{array}{rcl}\underline{s} & = & \underline{D : \bar{t}\Diamond\pi}\langle\langle x\rangle\underline{\pi'}\rangle\sigma_E \\ & = & \underline{D}\langle\langle\bar{t}(\langle x\rangle\underline{\pi'})\rangle\underline{\pi}\rangle\sigma_E \\ & = & \underline{D}\langle\langle\bar{t}\rangle\underline{x@\pi' : \pi}\rangle\sigma_E \quad = \quad \underline{s'}\end{array}$$

5. $s = (D, \lambda x.\bar{t}, \phi : \pi, E) \rightsquigarrow_{\mathtt{m}} (D, \bar{t}, \pi, [x{\leftarrow}\phi]E) = s'$. Then

$$\begin{array}{rcll} \underline{s} & = & \underline{D}\langle\langle\lambda x.\bar{t}\rangle\phi : \pi\rangle\sigma_E & \\ & = & \underline{D}\langle\langle(\lambda x.\bar{t})\phi\rangle\pi\rangle\sigma_E & \\ & \to_{\mathtt{r}\beta_f} & \underline{D}\langle\langle\bar{t}\{x{\leftarrow}\phi\}\rangle\pi\rangle\sigma_E & \\ & = & \underline{D}\langle\langle\bar{t}\rangle\pi\rangle\{x{\leftarrow}\phi\}\sigma_E & \\ & = & \underline{D}\langle\langle\bar{t}\rangle\pi\rangle\sigma_{[x\leftarrow\phi]E} & = \quad \underline{s'} \end{array}$$

where the rewriting step takes place because
(a) $\underline{D}\langle\pi\rangle\sigma_E$ is a right context by Lemma 10.4;
(b) ϕ is a fireball by Lemma 10.3.
Moreover, the meta-level substitution $\{x{\leftarrow}\phi\}$ can be extruded (in the equality step after the rewriting) without renaming x, because by Lemma 10.2 x does not occur in D nor π.

6. $$s = (D, x, \pi, E_1[x{\leftarrow}\lambda y.\overline{u}@\epsilon]E_2)$$
$$\rightsquigarrow_{\mathtt{e}} (D, (\lambda y.\overline{u})^{\alpha}, \pi, E_1[x{\leftarrow}\lambda y.\overline{u}@\epsilon]E_2) = s'.$$
Let $E' \coloneqq E_1[x{\leftarrow}\lambda y.\overline{u}@\epsilon]E_2$. Then

$$\begin{array}{rcll} \underline{s} & = & \underline{D}\langle\langle x\rangle\pi\rangle\sigma_{E'} & \\ & = & \underline{D\sigma_{E'}}\langle\langle x\sigma_{E'}\rangle\pi\sigma_{E'}\rangle & \\ & = & \underline{D\sigma_{E'}}\langle\langle\lambda y.\overline{u}\sigma_{E'}\rangle\underline{\pi\sigma_{E'}}\rangle & \\ & = & \underline{D}\langle\langle\lambda y.\overline{u}\rangle\pi\rangle\sigma_{E'} & = \quad \underline{s'} \end{array}$$

□

See p. 9 **Lemma 12** (Easy GLAMOUr Progress)**.** *Let s be a reachable final state. Then $\underline{s}$ is fireball, i.e. it is β_f-normal.*

Proof. An immediate inspection of the transitions shows that in a final state the code cannot be an application and the dump is necessarily empty. In fact, final states have one of the following two shapes:

1. *Top-Level Unapplied Abstraction*, *i.e.* $s = (\epsilon, \lambda x.\bar{t}, \epsilon, E)$. Then $\underline{s} = (\lambda x.\bar{t})\sigma_E = \lambda x.\bar{t}\sigma_E$ that is a fireball.
2. *Top-Level Free Variable or Inert Term with Free Head*, *i.e.* $s = (\epsilon, x, \pi, E)$ with $E(x) = \bot$. Then $\underline{s} = (\langle x\rangle\underline{\pi})\sigma_E = \langle x\sigma_E\rangle(\underline{\pi}\sigma_E) = \langle x\rangle(\underline{\pi}\sigma_E)$. Now, by the fireball item invariant (Lemma 10.3) every element of $\underline{\pi}\sigma_E$ is a fireball, and so $\langle x\rangle(\underline{\pi}\sigma_E)$ is a inert term, *i.e.* a fireball. □

See p. 9 **Theorem 4** (Weak Bisimulation)**.** *Let s be an initial state of code $\bar{t}$.*

1. Simulation: *For every execution ρ: $s \rightsquigarrow^* s'$ there exists a derivation d: $\underline{s} \to^*_{\mathtt{r}\beta_f} \underline{s'}$ such that $|d|_{\beta_f} = |\rho|_{\mathtt{m}}$;*
2. Reverse Simulation: *For every derivation d: $\bar{t} \to^*_{\mathtt{r}\beta_f} u$ there is an execution $\rho : s \rightsquigarrow^* s'$ such that $\underline{s'} = u$ and $|d|_{\beta_f} = |\rho|_{\mathtt{m}}$.*

Proof.

1. By induction on the length $|\rho|$ of ρ, using the one-step simulation lemma (Lemma 11). If ρ is empty then the empty derivation satisfies the statement. If ρ is given by $\sigma : s \rightsquigarrow^* s''$ followed by $s'' \rightsquigarrow s'$ then by *i.h.* there exists $e : \underline{s} \to^*_{\mathtt{r}\beta_f} \underline{s''}$ s.t. $|e| = |\sigma|_{\mathtt{m}}$. Cases of $s'' \rightsquigarrow s'$:
   (a) *Commutative or Exponential*. Then $\underline{s''} = \underline{s'}$ by Lemma 11.1 and the statement holds taking $d \coloneqq e$ because $|d|_{\beta_f} = |e| =_{i.h.} |\sigma|_{\mathtt{m}} = |\rho|_{\mathtt{m}}$.
   (b) *Multiplicative*. Then $\underline{s''} \to_{\mathtt{r}\beta_f} \underline{s'}$ by Lemma 11.2 and defining d as e followed by such a step we obtain $|d|_{\beta_f} = |e| + 1 =_{i.h.} |\sigma|_{\mathtt{m}} + 1 = |\rho|_{\mathtt{m}}$.
2. We use $\mathtt{nf}_{\mathtt{ec}}(s)$ to denote the normal form of s with respect to exponential and commutative transitions, that exists and is unique because $\rightsquigarrow_{\mathtt{c}} \cup \rightsquigarrow_{\mathtt{e}}$ terminates (termination is given by forthcoming Lemma 14 and Cor. 4, that are postponed because they actually give precise complexity bounds, not just termination) and the machine is deterministic (as it can be seen by an easy inspection of the transitions). The proof is by induction on the length of d. If d is empty then the empty execution satisfies the statement.
   If d is given by $e : \bar{t} \to^*_{\mathtt{r}\beta_f} s$ followed by $s \to_{\mathtt{r}\beta_f} u$ then by *i.h.* there is an execution $\sigma : s \rightsquigarrow^* s''$ s.t. $s = \underline{s''}$ and $|\sigma|_{\mathtt{m}} = |e|_{\beta_f}$. Note that since exponential and commutative transitions are mapped on equalities, σ can be extended as $\sigma' : s \rightsquigarrow^* s'' \rightsquigarrow^*_{\mathtt{e},\mathtt{c}_{1,2,3,4}} \mathtt{nf}_{\mathtt{ec}}(s'')$ with $\underline{\mathtt{nf}_{\mathtt{ec}}(s'')} = s$ and $|\sigma'|_{\mathtt{m}} = |e|_{\beta_f}$. By the progress property (Lemma 12) $\mathtt{nf}_{\mathtt{ec}}(s'')$ cannot be a final state, otherwise $s = \underline{\mathtt{nf}_{\mathtt{ec}}(s'')}$ could not reduce. Then $\mathtt{nf}_{\mathtt{ec}}(s'') \rightsquigarrow_{\mathtt{m}} s'$ (the transition is necessarily multiplicative because $\mathtt{nf}_{\mathtt{ec}}(s'')$ is normal with respect to the other transitions). By the one-step simulation lemma (Lemma 11.2) $\underline{\mathtt{nf}_{\mathtt{ec}}(s'')} = s \to_{\mathtt{r}\beta_f} \underline{s'}$ and by determinism of $\to_{\mathtt{r}\beta_f}$ (Lemma 9.2) $\underline{s'} = u$. Then the execution ρ defined as σ' followed by $\mathtt{nf}_{\mathtt{ec}}(s'') \rightsquigarrow_{\mathtt{m}} s'$ satisfy the statement, as $|\rho|_{\mathtt{m}} = |\sigma'|_{\mathtt{m}} + 1 = |\sigma|_{\mathtt{m}} + 1 = |e|_{\beta_f} + 1 = |d|_{\beta_f}$. □

Lemma 13 (Subterm Invariant)**.** *Let $s = (D, \bar{t}, \pi, E)$ be a state reachable from an initial code $\bar{t}_0$. If $\lambda x.\overline{u}$ is a subterm of D, $\bar{t}$, π, or E then it is a subterm of $\bar{t}_0$.* See p. 9

Proof. By induction on the length of the execution leading to the reachable state. In an initial state the invariant trivially holds. For a non-empty execution the proof is by a straightforward case analysis on the last transition, always relying on the *i.h.* □

Lemma 23 (Size Bounded)**.** *If $s = (D, \overline{u}, \pi, E)$ is a state reached by an execution ρ of initial code $\bar{t}$, then $|s|_c \leq (1 + |\rho|)|\bar{t}| - |\rho|_c$.*

Proof. By induction over the length of the derivation. The property trivially holds for the empty derivation. Case analysis over the last machine transition.

- *Commutative rule* $\rightsquigarrow_{\mathtt{c}_1}$: the rule splits the code $\bar{t}\overline{u}$ between the dump and the code, and the measure—as well as the rhs of the formula—decreases by 1 because the rule consumes the application node.
- *Commutative rules* $\rightsquigarrow_{\mathtt{c}_{2,3,4}}$: these rules consume the current code, so they decrease the measure of at least 1.
- *Multiplicative*: trivial, as the lhs decreases of 1 (because the λ of the abstraction is consumed) and the rhs does not change.
- *Exponential*: it modifies the current code by replacing a variable (of size 1) with an abstraction coming from the environment. Because of the subterm invariant (Lemma 13), the abstraction is a subterm of $\bar{t}$ and so the increment of $|s|_c$ is bounded by $|\bar{t}|$. □

Lemma 14 (Bilinearity of Commutative Transitions)**.** *For any state s reachable by an execution ρ of initial code $\bar{t}$, $|\rho|_c \leq (1 + |\rho|_{\mathtt{e}})|\bar{t}|$.* See p. 9

Proof. Immediate consequence of Lemma 23, since $|s|_c \geq 0$. □

Lemma 15 (Free Occurrences Invariant)**.** *Let $\rho : s \rightsquigarrow^* s'$ be an execution of initial code $\overline{u}$. Then $|s'|_{\mathtt{free}} \leq |\overline{u}|_{\mathtt{free}} + |\overline{u}| \cdot |\rho|_{\mathtt{m}} - |\rho|_{\mathtt{e}}$.* See p. 9

Proof. By induction on $|\rho|$. Case $|\rho| = 0$ is obvious. Otherwise $\sigma : s \rightsquigarrow^* s''$ and ρ extends σ with $s'' \rightsquigarrow s'$. By *i.h.*, $|s''|_{\mathtt{free}} \leq |\overline{u}|_{\mathtt{free}} + |\overline{u}| \cdot |\sigma|_{\mathtt{m}} - |\sigma|_{\mathtt{e}}$. Cases (the notation refers to the transitions of the machine, in Table 1):

- *the last transition is exponential.* We have to show $|s'|_{\mathtt{free}} \leq |\bar{t}|_{\mathtt{free}} + |\overline{u}| \cdot |\rho|_{\mathtt{m}} - |\rho|_{\mathtt{e}}$. It follows from the *i.h.* and

- $|s'|_{\mathsf{free}} = |s''|_{\mathsf{free}} - 1$ because dump and stack do not change and the code changes from a variable (of measure 1) to an abstraction (of measure 0);
- $|\rho|_{\mathsf{m}} = |\sigma|_{\mathsf{m}}$;
- $|\rho|_{\mathsf{e}} = |\sigma|_{\mathsf{e}} + 1$;

- *the last transition is multiplicative*. For $\rightsquigarrow_{\mathtt{m}}$:

$$\begin{aligned}
|\rho|_{\mathsf{free}} &= |D|_{\mathsf{free}} + |\pi|_{\mathsf{free}} + |\overline{t}|_{\mathsf{free}} \\
&\leq |D|_{\mathsf{free}} + |f : \pi|_{\mathsf{free}} + |\overline{t}|_{\mathsf{free}} && (|f|_{\mathsf{free}} \geq 0) \\
&= |D|_{\mathsf{free}} + |\lambda x.\overline{t}|_{\mathsf{free}} + |\lambda y.\overline{u} : \pi|_{\mathsf{free}} + |\overline{t}|_{\mathsf{free}} && (|\lambda x.\overline{t}|_{\mathsf{free}} = 0) \\
&= |s''|_{\mathsf{free}} + |\overline{t}|_{\mathsf{free}} && (\text{def. of } |s''|_{\mathsf{free}}) \\
&= |s''|_{\mathsf{free}} + |\overline{u}| && (\text{Lemma 13}) \\
&\leq |\overline{u}|_{\mathsf{free}} + |\overline{u}| \cdot |\sigma|_{\mathsf{m}} - |\sigma|_{\mathsf{e}} + |\overline{u}| && (\textit{i.h.}) \\
&= |\overline{u}|_{\mathsf{free}} + |\overline{u}| \cdot (|\sigma|_{\mathsf{m}} + 1) - |\sigma|_{\mathsf{e}} \\
&= |\overline{u}|_{\mathsf{free}} + |\overline{u}| \cdot |\rho|_{\mathsf{m}} - |\rho|_{\mathsf{e}}
\end{aligned}$$

- *the last transition is commutative*. Note that (sub)terms and stacks are moved around but never erased, never duplicated, and never modified. Moreover no new pieces of code are introduced, so that the measure never changes. Since also $|\rho|_{\mathsf{m}}$ and $|\rho|_{\mathsf{e}}$ do not change, the statement follows from the *i.h.* □

See p. 9 **Corollary 4** (Exponentials are Bilinear)**.** *Let s be an initial state of code $\overline{u}$ and $\rho : s \rightsquigarrow^* s'$. Then $|\rho|_{\mathsf{e}} \leq |\overline{u}| \cdot (|\rho|_{\mathsf{m}} + 1)$.*

Proof. By Lemma 15, $|\rho|_{\mathsf{e}} \leq |\overline{u}|_{\mathsf{free}} + |\overline{u}| \cdot |\rho|_{\mathsf{m}} - |s'|_{\mathsf{free}}$, that implies $|\rho|_{\mathsf{e}} \leq |\overline{u}|_{\mathsf{free}} + |\overline{u}| \cdot |\rho|_{\mathsf{m}}$. The statement follows from the fact that $|\overline{u}|_{\mathsf{free}} \leq |\overline{u}|$. □

See p. 9 **Theorem 5** (Easy GLAMOUr Overhead Bound)**.** *Let t be a term. Every derivation $d : t \to^*_{\mathtt{r}\beta_f} u$ is implementable on RAM in $O((1 + |d|_{\beta_f}) \cdot |t|^2)$, i.e. linear in the length of d and quadratic in the size of t.*

Proof. Given $d : t \to^*_{\mathtt{r}\beta_f} u$ by Thm. 4.2 there is an execution $\rho : s \rightsquigarrow^* s'$ such that $\underline{s} = t$, $\underline{s'} = u$, and $|\rho|_{\mathsf{m}} = |d|_{\beta_f}$. The cost of implementing ρ is the sum of the costs of implementing the multiplicative, exponential, and commutative transitions:

1. *Multiplicative*: each one costs $O(1)$ and so all together they cost $O(|d|_{\beta_f})$.
2. *Exponential*: by Cor. 4 we have $|\rho|_{\mathsf{e}} \leq (1 + |\rho|_{\mathsf{m}}) \cdot |t|$, *i.e.* the number of exponential transitions is bilinear. By the subterm invariant (Lemma 13), each exponential step costs at most $O(|t|)$, and so their full cost is $O((1 + |d|_{\beta_f}) \cdot |t|^2)$.
3. *Commutative*: by Lemma 14 we have $|\rho|_c \leq (1 + |\rho|_{\mathsf{e}})|t|$. Now, substituting the bound given by Cor. 4 we obtain

   $$|\rho|_c \leq (1+|\rho|_{\mathsf{e}})|t| \leq (1+(1+|\rho|_{\mathsf{m}})|t|)|t| = (1+|d|_{\beta_f}) \cdot |t|^2 + |t|$$

   Since every commutative transition evidently takes constant time, the whole cost of the commutative transitions is bound by $O((1 + |d|_{\beta_f}) \cdot |t|^2)$.

Then the cost of implementing ρ is $O((1 + |d|_{\beta_f}) \cdot |t|^2)$. □

B.6 Proofs of Section 7 (On the Minimality of the Cost Model)

See p. 10 **Proposition 9** (Exponentially many β_i-steps)**.** *For every $n \in \mathbb{N}$, one has $u_n \to^{2^n-1}_{\beta_i} r_n$.*

Proof. By induction on n. For the base case $n = 0$ we have $u_0 = r_0$ by definition. For the inductive case,

$$\begin{aligned}
u_{n+1} = t_{n+1}i = (\lambda z.(yu_nu_n))i &\to_{\beta_i} yu_nu_n \\
&\to^{2^n-1}_{\beta_i} yr_nu_n \to^{2^n-1}_{\beta_i} yr_nr_n = r_{n+1}
\end{aligned}$$

where the two $\to^{2^n-1}_{\beta_i}$ sequence are obtained by *i.h.*. We have $u_{n+1} \to^{1+2*(2^n-1)}_{\beta_i} r_{n+1}$, with $1 + 2 * (2^n - 1) = 2^{n+1} - 1$. □

Proposition 10 (Linearly many β_v-steps)**.** *For every $n \in \mathbb{N}$, one has $s_n \to^n_{\beta_v} t_n$, and so $s_ni \to^n_{\beta_v} t_ni = u_n$.* See p. 10

Proof. By induction on n. For the base case $n = 0$ we have $s_0 = t_0$ by definition. For the inductive case, since all t_n's are values, we obtain

$$\begin{aligned}
s_{n+1} = (\lambda x.\lambda z.(y(xi)(xi)))s_n &\to^n_{\beta_v} (\lambda x.\lambda z.(y(xi)(xi)))t_n \\
&\to_{\beta_v} \lambda z.(y(t_ni)(t_ni)) = \lambda z.(yu_nu_n) = t_{n+1}
\end{aligned}$$

Note that the n-th β_v-step duplicates t_n, and that t_n is an abstraction for $n > 1$, *i.e.* the only variable step in $s_n \to^n_{\beta_v} t_n$ is the first one. In other words, variable steps do not play a role here. □