On the Relative Usefulness of Fireballs

Beniamino Accattoli
INRIA & LIX/École Polytechnique

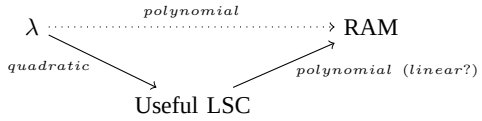
Claudio Sacerdoti Coen
University of Bologna

***Abstract*—In CSL-LICS 2014, Accattoli and Dal Lago [1] showed that there is an implementation of the ordinary (i.e. strong, pure, call-by-name) λ-calculus into models like RAM machines which is polynomial in the number of β-steps, answering a long-standing question. The key ingredient was the use of a calculus with useful sharing, a new notion whose complexity was shown to be polynomial, but whose implementation was not explored. This paper, meant to be complementary, studies useful sharing in a call-by-value scenario and from a practical point of view. We introduce the *Fireball Calculus*, a natural extension of call-by-value to open terms, that is an intermediary step towards the strong case, and we present three results. First, we adapt and refine useful sharing, refining the meta-theory. Then, we introduce the GLAMoUr a simple abstract machine implementing the Fireball Calculus extended with useful sharing. Its key feature is that usefulness of a step is tested—surprisingly—in constant time. Third, we provide a further optimisation that leads to an implementation having only a linear overhead with respect to the number of β-steps.**

I. Introduction

The λ-calculus is an interesting computational model because it is machine-independent, simple to define, and it compactly models functional and higher-order logic programming languages. Its definition has only one rule, the β rule, and no data structures. The catch is the fact that the β-rule—which by itself is Turing-complete—is not an atomic rule. Its action, namely $(\lambda x.t)u \to_\beta t\{x \leftarrow u\}$, can make many copies of an arbitrarily big sub-program u. In other computational models like Turing or RAM machines, an atomic operation can only move the head on the ribbon or access a register. Is β atomic in that sense? Can one count the number of β-steps to the result and then claim that it is a reasonable bound on the complexity of the term? Intuition says no, because β can be nasty, and make the program grow at an exponential rate. This is the *size explosion problem*.

Useful Sharing: nonetheless, it is possible to take the number of β-steps as an invariant cost model, *i.e.* as a complexity measure polynomially related to RAM or Turing machines. While this was known for some notable sub-calculi [2]–[6], the first proof for the general case is a recent result by Accattoli and Dal Lago [1]. Similarly to the literature, they circumvent size explosion by factoring the problem via an intermediary model in between λ-calculus and machines. Their model is the *linear substitution calculus* (LSC) [1], [7], that is a simple λ-calculus with sharing annotations (also known as explicit substitutions) where the substitution process is decomposed in micro steps, replacing one occurrence at a time. In contrast with the literature, the general case is affected by a stronger form of size explosion, requiring an additional and sophisticated layer of sharing, called *useful sharing*. Roughly, a micro substitution step is *useful* if it contributes somehow to the creation of a β-redex, and *useless* otherwise. Useful reduction then selects only useful substitution steps, avoiding the useless ones. In [1], the Useful LSC is shown to be polynomially related to both l-calculus (in a quadratic way) and RAM machines (with polynomial overhead, conjectured linear). It therefore follows that there is a polynomial relationship $\lambda \to$ RAM. Pictorially:



Coming back to our questions, the answer of [1] is yes, β is atomic, up to a polynomial overhead. It is natural to wonder how big this overhead is. Is β reasonably atomic? Or is the degree of the polynomial big and does the invariance result only have a theoretical value? In particular, in [1] the definition of useful steps relies on a *separate* and *global* test for usefulness, that despite being tractable might not be feasible in practice. Is there an efficient way to implement the Useful LSC? Does useful sharing—*i.e.* the avoidance of useless duplications—bring a costly overhead? This paper answers these questions. But, in order to stress the practical value of the study, it shifts to a slightly different setting.

The Fireball Calculus: we recast the problem in terms of the new *fireball calculus* (FBC), essentially the weak call-by-value l-calculus generalised to handle open terms. It is an intermediary step towards a strong call-by-value l-calculus, that can be seen as iterated open weak evaluation. A similar approach to strong evaluation is followed also by Grégoire and Leroy in [8]. It avoids some of the complications of the strong case, and at the same time exposes all the subtleties of dealing with open terms.

Free variables are actually formalised using a distinguished syntactic class, that of *symbols*, noted a, b, c. This approach is technically convenient because it allows restricting to closed terms, so that any variable occurrence x is bound somewhere, while still having a representation of free variables (as symbols).

The basic idea is that—in the presence of symbols—restricting β-redex to *fire* only in presence of values is problematic. Consider indeed the following term:

$$t := ((\lambda x.\lambda y.u)(aa))w$$

where w is normal. For the usual call-by-value operational semantics t is normal (because aa is not a value) while for theoretical reasons (see [9]–[11]) one would like to be able to fire the blocked redex, reducing to $(\lambda y.u\{x{\leftarrow}aa\})w$, so that a new redex is created and the computation can continue. According to the standard classification of redex creations due to Lévy [12], this is a creation of type 1[1].

The solution we propose here is to relax the constraint about values, allowing β-redexes to fire whenever the argument is a more general structure, a so-called *fireball*, defined recursively by extending values with *inerts*, *i.e.* applications of symbols to fireballs. In particular, aa is inert, so that *e.g.* $t \to (\lambda y.u\{x{\leftarrow}aa\})w$, as desired.

Functional programming languages are usually modelled by weak and *closed* calculi, so it is natural to wonder about the practical relevance of the FBC. Applications are along two axes. On the one hand, the evaluation mechanism at work in proof assistants has to deal with open terms for comparison and unification. For instance, Gregoire and Leroy's [8], meant to improve the implementation of Coq, relies on inerts (therein called *accumulators*). On the other hand, symbols may also be interpreted as *constructors*, meant to represent data as lists or trees. The dynamics of fireballs is in fact consistent with the way constructors are handled by Standard ML [13] and in several formalisation of core ML, as in [14]. In this paper we omit destructors, whose dynamics is orthogonal to the one of β-reduction, and we expect all results presented here to carry-over with minor changes to a calculus with destructors. Therefore firing redexes involving inerts is also justified from a practical perspective.

The Relative Usefulness of Fireballs: as we explained, the generalisation of values to fireballs is motivated by creations of type 1 induced by the firing of inerts. There is a subtlety, however. While *substituting* a value can create a new redex (*e.g.* as in $(\lambda x.(xI))I \to (xI)\{x{\leftarrow}I\} = II$, where I is the identity—these are called creations of type 3), substituting a inert can not. Said differently, duplicating inerts is useless, and leads to size explosion. Note the tension between different needs: redexes involving inerts have to be fired (for creations of type 1), and yet the duplication and the substitution of inerts should be avoided (since they do not give rise to creations of type 3). We solve the tension by turning to sharing, and use the simplicity of the framework to explore the implementation of useful sharing. Both values and inerts (*i.e.* fireballs) in argument position will trigger reduction, and both will be shared after the redex is reduced, but only the substitution of values might be useful, because inerts are always useless. This is what we call *the relative usefulness of fireballs*. It is also why—in contrast to Gregoire and Leroy—we do not identify fireballs and values.

[1]The reader unfamiliar with redex creations should not worry. Creations are a key concept in the study of usefulness—which is why we mention them—but for the present discussion it is enough to know that there exists two kinds of creations (type 1 and the forthcoming type 3, other types will not play a role), no expertise on creations is required.

The Result: our main result is an implementation of FBC relying on useful sharing and such that it has only a linear overhead with respect to the number of β-steps. To be precise, the overhead is *bilinear*, *i.e.* linear in the number of β-steps *and* in the size of the initial term (roughly the size of the input). The dependency from the size of the initial term is induced by the action of β on whole subterms, rather than on atomic pieces of data as in RAM or Turing machines. Therefore, β is not exactly as atomic as accessing a register or moving the head of a Turing machine, and this is the price one must pay for embracing higher-order computations. Bilinearity, however, guarantees that such a price is mild and that the number of β step—*i.e.* of function calls in a functional program—is a faithful measure of the complexity of a program. To sum up, our answer is yes, β is also reasonably atomic.

A Recipe for Bilinearity, with Three Ingredients: our proof technique is a *tour de force* progressively combining together and adapting to the FBC three recent works involving the LSC, namely the already cited invariance of useful sharing of [1], the tight relationship with abstract machines developed by Accattoli, Barenbaum, and Mazza in [15], and the optimisation of the substitution process studied by the present authors in [16]. The next section will give an overview of these works and of how they are here combined, stressing how the proof is more than a simple stratification of techniques. In particular, it was far from evident that the orthogonal solutions introduced by these works could be successfully combined together.

This Paper: the paper is meant to be self-contained, and mostly follows a didactic style. For the first half we warm up by discussing design choices, the difficulty of the problem, and the abstract architecture. The second half focuses on the results. We also suggest reading the introductions of [1], [15], [16], as they provide intuitions about concepts that here are only hinted at. Although not essential, they will certainly soften the reading of this work. Omitted proofs are in the appendix and related work is discussed in Sect. III.

II. A Recipe with Three Ingredients

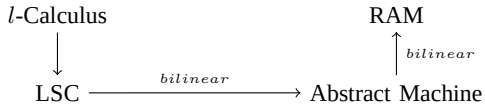
This section gives a sketch of how the bilinear implementation is built by mixing together tools from three different studies on the LSC.

1) Useful Fireballs: we start by introducing the Useful Fireball Calculus (Useful FBC), akin to the Useful LSC, and provide the proof that the relationship FBC $\to$ Useful FBC, analogously to the arrow $l \to$ Useful LSC, has a quadratic overhead. Essentially, this step provides us with the following diagram:

We go beyond simply adapting the study of [1], as the use of evaluation contexts (typical of call-by-value scenarios) leads to the new notion of *useful evaluation context*, that simplifies the technical study of useful sharing. Another key point is the

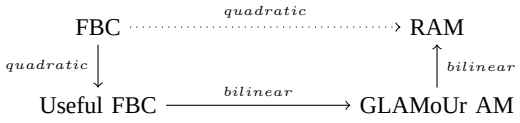
relative usefulness of fireballs, according to their nature: only values are properly subject to the useful discipline, *i.e.* are duplicated only when they contribute somehow to β-redexes, while inerts are never duplicated.

2) Distilling Useful Fireballs: actually, we do not follow [1] for the study of the arrow Useful FBC $\to$ RAM. We rather refine the whole picture, by introducing a further intermediary model, an *abstract machine*, mediating between the Useful FBC and RAM. We adopt the *distillation technique* of [15], that establishes a fine-grained and modular view of abstract machines as strategies in the LSC up to a notion of structural equivalence on terms. The general pattern arising from [15] is that for call-by-name/value/need weak and closed calculi the abstract machine adds only a bilinear overhead with respect to the shared evaluation within the LSC:



Such *distilleries* owe their name to the fact that the LSC retains only part of the dynamics of a machine. Roughly, it isolates the relevant part of the computation, distilling it away from the search for the next redex implemented by abstract machines. The search for the redex is mapped to a notion of structural equivalence, a particular trait of the LSC, whose key property is that it can be postponed. Additionally, the transitions implementing the search for the next redex are proved to be bilinear in those simulated by the LSC: the LSC then turns out to be a complexity-preserving abstraction of abstract machines.

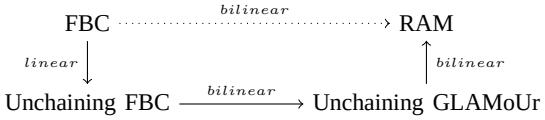
The second ingredient for the recipe is then a new abstract machine, called GLAMoUr, that we prove implements the Useful FBC within a bilinear overhead. Moreover, the GLAMoUr itself can be implemented within a bilinear overhead. Therefore, we obtain the following diagram:



This is the most interesting and original step of our study. First, it shows that distilleries are compatible with open terms and useful sharing. Second, while in [15] distilleries were mainly used to revisit machines in the literature, here the distillation principles are used to guide the design of a new abstract machine. Third, useful sharing is handled via a refinement of an ordinary abstract machine relying on a basic form of labelling. The most surprising fact is that such a labelling (together with invariants induced by the call-by-value scenario) allows a straightforward and very efficient implementation of useful sharing. While the calculus is based on *separate* and *global* tests for the usefulness of a substitution step, the labelling allows the machine to do *on-the-fly* and *local* tests, requiring only constant time (!). It then turns out that implementing usefulness is much easier than analysing it. Summing up, useful sharing is easy to implement and thus a remarkable case of a theoretically born concept with relevant practical consequences.

3) Unchaining Substitutions: at this point, it is natural to wonder if the bottleneck given by the side of the diagram FBC $\to$ Useful FBC, due to the overhead of the decomposition of substitutions, can be removed. The bound on the overhead is in fact tight, and yet the answer is yes, if one refines the actors of the play. Our previous work [16], showed that (in ordinary weak and closed settings) the quadratic overhead is due to malicious chains of *renamings*, *i.e.* of substitutions of variables for variables, and that the substitution overhead reduces to linear if the evaluation is modified so that variables are never substituted, *i.e.* if values do not include variables.

For the fireball calculus the question is tricky. First of all a disclaimer: with *variables* we refer to occurrences of bound variables and not to symbols/free variables. Now, our initial definition of the calculus will exclude variables from fireballs, but useful sharing will force us to somehow reintroduce them. Our way out is an optimised form of substitution that *unchains* renaming chains, and whose overhead is proved linear by a simple amortised analysis. Such a third ingredient is first mixed with both the Useful FBC and the GLAMoUr, obtaining the Unchaining FBC and the Unchaining GLAMoUr, and then used to prove our main result, an implementation FBC $\to$ RAM having an overhead linear in the number of β steps and in the size of the initial term:



In this step, the original content is that the unchaining optimisation—while inspired by [16]—is subtler to define than in [16], as bound variables cannot be simply removed from the definition of fireballs, because of usefulness. Moreover, we also show how such an optimisation can be implemented at the machine level.

The next section discusses related work. Then there will be a long preliminary part providing basic definitions, an abstract decomposition of the implementation, and a quick study of both a calculus, the Explicit FBC, and a machine, the GLAM, without useful sharing. Both the calculus and the machine will not have any good asymptotical property, but they will be simple enough to familiarise the reader with the framework and with the many involved notions.

III. Related Work

In the literature, invariance results for the weak call-by-value l-calculus have been proved three times, independently. First, by Blelloch and Greiner [2], while studying cost models for parallel evaluation. Then by Sands, Gustavsson and Moran [3], while studying speedups for functional languages, and finally by Dal Lago and Martini [4], who addressed the invariance thesis for l-calculus. Among them, [3] is the closest one, as it also provides an abstract machine and bounds its overhead. These works however concern closed terms, and so

they deal with a much simpler case. Other simple call-by-name cases are studied in [5] and [6]. The difficult case of the strong l-calculus has been studied in [1], which is also the only reference for useful sharing.

The LSC is a variation over a l-calculus with ES by Robin Milner [17], [18], obtained by plugging in some of the ideas of the structural l-calculus by Accattoli and Kesner [19], introduced as a syntactic reformulation of linear logic proof nets. The LSC is similar to calculi studied by De Bruijn [20] and Nederpelt [21]. Its first appearances in the literature are in [6], [22], but its inception is actually due to Accattoli and Kesner.

Many abstract machines can be rephrased as strategies in *l-calculi with explicit substitutions* (ES), see at least [23]–[28].

The related work that is by far closer to ours is the already cited study by Grégoire and Leroy of an abstract machine for call-by-value weak and open reduction in [8]. We developed our setting independently, and yet the FBC is remarkably close to their calculus, in particular our *inerts* are essentially their *accumulators*. The difference is that our work is complexity-oriented while theirs is implementation-oriented. On the one hand they do not recognise the relative usefulness of fireballs, and so their machine is not invariant, *i.e.* our machine is more efficient and on some terms even exponentially faster. On the other hand, they extend the language up to the calculus of constructions, present a compilation to bytecode, and certify in Coq the correctness of the implementation.

The abstract machines in this paper use *global* environments, an approach followed only by a minority of authors (*e.g.* [3], [15], [29], [30]) and essentially identifying the environment with a store.

The distillation technique was developed to better understand the relationship between the KAM and weak linear head reduction pointed out by Danos & Regnier [31]. The idea of distinguishing between *operational content* and *search for the redex* in an abstract machine is not new, as it underlies in particular the *refocusing semantics* of Danvy and Nielsen [32]. Distilleries however bring an original refinement where logic, rewriting, and complexity enlighten the picture, leading to formal bounds on machine overheads.

Our unchaining optimisation is a lazy variant of an optimisation that repeatedly appeared in the literature on abstract machines, often with reference to space consumption and *space leaks*, for instance in [3] as well as in Wand's [33] (section 2), Friedman et al.'s [34] (section 4), and Sestoft's [35] (section 4).

IV. The Fireball Calculus

The setting is the one of the call-by-value λ-calculus extended with symbols a, b, c, meant to denote free variables (or constructors). The syntax is:

$$\begin{array}{lllll} \text{Terms} & t,u,w,r & ::= & x \mid a \mid lx.t \mid tu \\ \text{Values} & v,v' & ::= & lx.t \end{array}$$

with the usual notions of free and bound variables, capture-avoiding substitution $t\{x{\leftarrow}u\}$, and closed (*i.e.* without free variables) term. We will often restrict to consider closed terms, the idea being that an open term as $x(\lambda y.zy)$ is rather represented as the closed term $a(\lambda y.by)$.

The ordinary (*i.e.* without symbols) call-by-value l-calculus has a nice operational characterisation of values:

closed normal forms are values

Now, the introduction of symbols breaks this property, because there are closed normal forms as $a(\lambda x.x)$ that are not values. In order to restore the situation, we generalise values to *fireballs*[2], that are either values v or *inerts* A, *i.e.* symbols possibly applied to fireballs. Associating to the left, fireballs and inerts are compactly defined by

$$\begin{array}{lllll} \text{Fireballs} & f,g,h & ::= & v \mid A \\ \text{Inerts} & A,B,C & ::= & af_1 \dots f_n \quad n \geq 0 \end{array}$$

For instance, $\lambda x.y$ and a are fireballs, as well as $a(\lambda x.x)$, ab, and $(a(\lambda x.x))(bc)(\lambda y.(zy))$. Fireballs can also be defined more atomically by mixing values and inerts as follows:

$$f \quad ::= \quad v \mid A \qquad\qquad A \quad ::= \quad a \mid Af$$

Note that AB and AA are always inerts.

Next, we generalise the call-by-value rule $(\lambda x.t)v \to_{\beta_v} t\{x{\leftarrow}v\}$ to substitute fireballs f rather than values v. First of all, we define a notion of evaluation context (noted F rather than E, reserved to forthcoming global environments), mimicking right-to-left CBV evaluation:

$$\text{Evaluation Contexts} \quad F \quad ::= \quad \langle\cdot\rangle \mid tF \mid Ff$$

note the case Ff, that in CBV would be Fv. Last, we define the $\mathtt{f}$(fireball) rule $\to_{\mathtt{f}}$ as follows

$$\begin{array}{ll} \textsc{Rule at Top Level} & \textsc{Contextual closure} \\ (lx.t)f \mapsto_{\mathtt{f}} t\{x{\leftarrow}f\} & F\langle t\rangle \to_{\mathtt{f}} F\langle u\rangle \quad \text{if } t \mapsto_{\mathtt{f}} u \end{array}$$

Our definitions lead to:

Theorem 1.

1) *Closed normal forms are fireballs.*
2) $\to_{\mathtt{f}}$ *is deterministic.*

In the introduction we motivated the notion of fireball both from theoretical and practical points of view. Theorem 1.1 provides a further, strong justification: it expresses a sort of internal harmony of the FBC, allowing to see it as the canonical completion of call-by-value to the open setting.

V. Size Explosion

Size Explosion is the side effect of a discrepancy between the dynamics and the representation of terms. The usual substitution $t\{x{\leftarrow}u\}$ makes copies of u for all the occurrences of x, even if u is *useless*, *i.e.* it is normal and it does not create redexes after substitution. These copies are the burden leading to the exponential growth of the size. To illustrate the problem, let's build a size exploding family of terms.

[2]About *fireball*: the first choice was *fire-able*, but then the spell checker suggested *fireball*.

Table I
SYNTAX, REWRITING RULES, AND STRUCTURAL EQUIVALENCE OF THE EXPLICIT FBC

$$
\begin{array}{lcl}
t,u,w,r & ::= & x \mid a \mid lx.t \mid tu \mid t[x{\leftarrow}u] \\
v,v' & ::= & lx.t \\
L,L' & ::= & \langle\cdot\rangle \mid L[x{\leftarrow}t] \\
A,B,C & ::= & a \mid L\langle A\rangle L\langle f\rangle \\
f,g,h & ::= & v \mid A \\
F & ::= & \langle\cdot\rangle \mid tF \mid FL\langle f\rangle \mid F[x{\leftarrow}t]
\end{array}
$$

RULE AT TOP LEVEL	CONTEXTUAL CLOSURE
$L\langle lx.t\rangle L'\langle f\rangle \mapsto_{\mathtt{m}} L\langle t[x{\leftarrow}L'\langle f\rangle]\rangle$	$F\langle t\rangle \multimap_{\mathtt{m}} F\langle u\rangle$ if $t \mapsto_{\mathtt{m}} u$
$F\langle x\rangle[x{\leftarrow}L\langle f\rangle] \mapsto_{\mathtt{e}} L\langle F\langle f\rangle[x{\leftarrow}f]\rangle$	$F\langle t\rangle \multimap_{\mathtt{e}} F\langle u\rangle$ if $t \mapsto_{\mathtt{e}} u$

$$
\begin{array}{rcll}
t[x{\leftarrow}u][y{\leftarrow}w] & \equiv_{com} & t[y{\leftarrow}w][x{\leftarrow}u] & \text{if } y \notin \mathtt{fv}(u) \text{ and } x \notin \mathtt{fv}(w) \\
(tw)[x{\leftarrow}u] & \equiv_{@r} & tw[x{\leftarrow}u] & \text{if } x \notin \mathtt{fv}(t) \\
(tw)[x{\leftarrow}u] & \equiv_{@l} & t[x{\leftarrow}u]w & \text{if } x \notin \mathtt{fv}(w) \\
t[x{\leftarrow}u][y{\leftarrow}w] & \equiv_{[\cdot]} & t[x{\leftarrow}u[y{\leftarrow}w]] & \text{if } y \notin \mathtt{fv}(t)
\end{array}
$$

Note that a inert A, when applied to itself still is a inert AA. In particular, it still is a fireball, and so it can be used as an argument for redexes. We can then easily build a term of size linear in n that in n steps evaluates a complete binary tree A^{2^n}. Namely, define the family of terms t_n for $n \geq 1$:

$$
\begin{array}{lcl}
t_1 & := & \lambda x_1.(x_1x_1) \\
t_{n+1} & := & \lambda x_{n+1}.(t_n(x_{n+1}x_{n+1}))
\end{array}
$$

Now consider t_nA, that for a fixed A has size linear in n. The next proposition shows that t_nA reduces in n steps to A^{2^n}, causing size-explosion.

Proposition 1 (Size Explosion in the FBC)**.** $t_nA \to_{\mathtt{f}}^n A^{2^n}$.

Proof: by induction on n. Let $B := A^2 = AA$. Cases:

$$
\begin{array}{lcll}
t_1 & = & (\lambda x_1.(x_1x_1))A & \to_{\mathtt{f}} \quad A^2 \\
t_{n+1} & = & (\lambda x_{n+1}.(t_n(x_{n+1}x_{n+1})))A & \to_{\mathtt{f}} \\
 & & t_nA^2 = t_nB & \to_{\mathtt{f}}^n \qquad (\textit{i.h.}) \\
 & & B^{2^n} & = \quad A^{2^{n+1}} \qquad \blacksquare
\end{array}
$$

VI. FIREBALLS AND EXPLICIT SUBSTITUTIONS

In a ordinary weak scenario, sharing of subterms prevents size explosion. In the FBC however this is no longer true, as we show in this section. Sharing of subterms is here represented in a variation over the Linear Substitution Calculus, a formalism with explicit substitutions coming from a linear logic interpretation of the λ-calculus. At the dynamic level, the *small-step* operational semantics of the FBC is refined into a *micro-step* one, where explicit substitutions replace one variable occurrence at a time, similarly to abstract machines.

The language of the *Explicit Fireball Calculus* (Explicit FBC) is:

$$t,u,w,r \quad ::= \quad x \mid a \mid lx.t \mid tu \mid t[x{\leftarrow}u]$$

where $t[x{\leftarrow}u]$ is the explicit substitution (ES) of u for x in t, that is an alternative notation for $\mathtt{let}\ x = u\ \mathtt{in}\ t$, and where x becomes bound (in t). We silently work modulo α-equivalence of these bound variables, *e.g.* $(xy)[y{\leftarrow}t]\{x{\leftarrow}y\} = (yz)[z{\leftarrow}t]$. We use $\mathtt{fv}(t)$ for the set of free variables of t.

Contexts: the dynamics of explicit substitutions is defined using (one-hole) contexts. *Weak contexts* subsume all the kinds of context in the paper, and are defined by

$$W,W' \quad ::= \quad \langle\cdot\rangle \mid tW \mid Wt \mid W[x{\leftarrow}t] \mid t[x{\leftarrow}W]$$

The plugging $W\langle t\rangle$ of a term t into a context W is defined as $\langle\cdot\rangle\langle t\rangle := t$, $(lx.W)\langle t\rangle := lx.(W\langle t\rangle)$, and so on. As usual, plugging in a context can capture variables, *e.g.* $((\langle\cdot\rangle y)[y{\leftarrow}t])\langle y\rangle = (yy)[y{\leftarrow}t]$. The plugging $W\langle W'\rangle$ of a context W' into a context W is defined analogously. Since all kinds of context we will deal with will be weak, the definition of plugging applies uniformly to all of them.

A special and frequently used class of contexts is that of *substitution contexts* $L ::= \langle\cdot\rangle \mid L[x{\leftarrow}t]$.

Switching from the FBC to the Explicit FBC the syntactic categories of *inerts* A, *fireballs* f, and *evaluation contexts* F are generalised in Table I as to include substitution contexts L. Note that fireballs may now contain substitutions, but *not at top level*, because it is technically convenient to give a separate status to a fireball f in a substitution context L: terms of the form $L\langle f\rangle$ are called *answers*. An *initial term* is a closed term with no explicit substitutions.

Rewriting Rules: the fireball rule $\to_{\mathtt{f}}$ is replaced by $\multimap_{\mathtt{f}}$, defined as the union of the two rules $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ in Table I:

1) *Multiplicative* $\multimap_{\mathtt{m}}$: is a version of $\to_{\mathtt{f}}$ where $lx.t$ and f can have substitution contexts L and L' around, and the substitution is delayed.
2) *Exponential* $\multimap_{\mathtt{e}}$: the substitution or exponential rule $\multimap_{\mathtt{e}}$ replaces exactly one occurrence of a variable x currently under evaluation (in F) with its definiendum f given by the substitution. Note the apparently strange position of L in the reduct. It is correct: L has to commute outside to bind both copies of f, otherwise the rule would create free variables.

The name of the rules are due to the linear logic interpretation of the LSC.

Unfolding: the shared representation is related to the usual one via the crucial notion of *unfolding*, producing the l-term $t{\downarrow}$ denoted by t and defined by:

$$
\begin{array}{lcllcl}
x{\downarrow} & := & x & (tu){\downarrow} & := & t{\downarrow}u{\downarrow} \\
(lx.t){\downarrow} & := & lx.t{\downarrow} & t[x{\leftarrow}u]{\downarrow} & := & t{\downarrow}\{x{\leftarrow}u{\downarrow}\}
\end{array}
$$

Note that $r_n{\downarrow} = A^{2^n}$.

As for the FBC, evaluation is well-defined:

Theorem 2.

1) *Closed normal forms are answers, i.e. fireballs in substitution contexts.*
2) $\multimap_{\mathtt{f}}$ *is deterministic.*

Structural Equivalence: the calculus is endowed with a structural equivalence, noted $\equiv$, whose property is to be a strong bisimulation with respect to $\multimap_{\mathtt{f}}$. It is the least equivalence relation closed by weak contexts defined in Table I.

Proposition 2 ($\equiv$ is a Strong Bisimulation wrt $\multimap_{\mathtt{f}}$)**.** *Let* $\mathtt{x} \in \{\mathtt{sm}, \mathtt{se}\}$. *Then,* $t \equiv u$ *and* $t \multimap_{\mathtt{x}} t'$ *implies that there exists* u' *such that* $u \multimap_{\mathtt{x}} u'$ *and* $t' \equiv u'$.

Size Explosion, Again: coming back to the size explosion example, the idea is that—to circumvent it—t_n should better $\multimap_{\mathtt{m}}$-evaluate to:

$$r_n := (x_0x_0)[x_0 \leftarrow x_1^2][x_1 \leftarrow x_2^2] \dots [x_{n-1} \leftarrow x_n^2][x_n \leftarrow A]$$

which is an alternative, compact representation of A^{2^n}, of size linear in n, and with just one occurrence of A. Without symbols, ES are enough to circumvent size explosion [2]–[4]. In our case however they fail. The evaluation we just defined indeed does not stop on the desired compact representation, and in fact a linear number of steps (namely $3n$) may still produce an exponential output (in a substitution context).

Proposition 3 (Size Explosion in the Explicit FBC)**.** $t_n A(\multimap_{\mathtt{m}}\multimap_{\mathtt{e}}^2)^n L\langle A^{2^n}\rangle$.

Proof: by induction on n. Let $B := A^2 = AA$. Cases:

$$\begin{array}{rcll} t_1 & = & (\lambda x_1.(x_1x_1))A & \multimap_{\mathtt{m}} \\ & & (x_1x_1)[x_1 \leftarrow A] & \multimap_{\mathtt{e}} \\ & & (x_1A)[x_1 \leftarrow A] & \multimap_{\mathtt{e}} \\ & & (AA)[x_1 \leftarrow A] & = \quad A^2[x_1 \leftarrow A] \\ t_{n+1} & = & (\lambda x_{n+1}.(t_n(x_{n+1}x_{n+1})))A & \multimap_{\mathtt{m}}\multimap_{\mathtt{e}}^2 \\ & & (t_nA^2)[x_1 \leftarrow A] = L\langle t_nB\rangle & (\multimap_{\mathtt{m}}\multimap_{\mathtt{e}}^2)^n \text{ (i.h.)} \\ & & L'\langle B^{2^n}\rangle = L'\langle A^{2^{n+1}}\rangle & \blacksquare \end{array}$$

Before introducing useful evaluation—that will liberate us from size explosion—we are going to fully set up the architecture of the problem, by explaining 1) how ES implement a calculus, 2) how an abstract machine implements a calculus with ES, and 3) how to define an abstract machine for the inefficient Explicit FBC. Only by then (Sect. XI) we will start optimising the framework, first with useful sharing and then by eliminating renaming chains.

VII. Two Levels Implementation

Here we explain how the the small-step strategy $\to_{\mathtt{f}}$ of the FBC is implemented by a micro-step strategy $\multimap$. We are looking for an appropriate strategy $\multimap$ with ES which is polynomially related to both $\to_{\mathtt{f}}$ and an abstract machine. Then we need two theorems:

1) *High-Level Implementation*: $\to_{\mathtt{f}}$ terminates iff $\multimap$ terminates. Moreover, $\to_{\mathtt{f}}$ is implemented by $\multimap$ with only a polynomial overhead. Namely, $t \multimap^k u$ iff $t \to_{\mathtt{f}}^h u{\downarrow}$ with k polynomial in h;
2) *Low-Level Implementation*: $\multimap$ is implemented on an abstract machine with an overhead in time which is polynomial in both k and the size of t.

We will actually be more accurate, giving linear or quadratic bounds, but this is the general setting.

A. High-Level Implementation

First, terminology and notations. *Derivations* $d, e, \dots$ are sequences of rewriting steps. With $|d|$, $|d|_{\mathtt{m}}$, and $|d|_{\mathtt{e}}$ we denote respectively the length, the number of multiplicative, and exponential steps of d.

Definition 1. *Let* $\to_{\mathtt{f}}$ *be a deterministic strategy on FBC-terms and* $\multimap$ *a deterministic strategy for terms with ES. The pair* $(\to_{\mathtt{f}}, \multimap)$ *is a* ***high-level implementation system*** *if whenever* t *is a* l*-term and* $d : t \multimap^* u$ *then:*

1) Normal Form*: if* u *is a* $\multimap$*-normal form then* $u{\downarrow}$ *is a* $\to_{\mathtt{f}}$*-normal form.*
2) Projection*:* $d{\downarrow} : t{\downarrow} \to_{\mathtt{f}}^* u{\downarrow}$ *and* $|d{\downarrow\!\!\downarrow}| = |d|_{\mathtt{m}}$.

Moreover, it is

1) locally bounded*: if the length of a sequence of substitution* $\mathtt{e}$*-steps from* u *is linear in the number* $|d|_{\mathtt{m}}$ *of* $\mathtt{m}$*-steps in* d*;*
2) globally bounded*: if* $|d|_{\mathtt{e}}$ *is linear in* $|d|_{\mathtt{m}}$.

The normal form and projection properties address the *qualitative* part, *i.e.* the part about termination. The normal form property guarantees that $\multimap$ does not stop prematurely, so that when $\multimap$ terminates $\to_{\mathtt{f}}$ cannot keep going. The projection property guarantees that termination of $\to_{\mathtt{f}}$ implies termination of $\multimap$. The two properties actually state a stronger fact: $\to_{\mathtt{f}}$ *steps can be identified with the* $\multimap_{\mathtt{m}}$*-steps of the* $\multimap$ *strategy*.

The local and global bounds allow to bound the overhead introduced by the Explicit FBC wrt the FBC, because by relating $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ steps, they relate $|d|$ and $|d{\downarrow\!\!\downarrow}|$, since $\to_{\mathtt{f}}$ and $\multimap_{\mathtt{m}}$ steps can be identified.

The high-level part can now be proved abstractly.

Theorem 3 (High-Level Implementation)**.** *Let* t *be an ordinary* l*-term and* $(\to_{\mathtt{f}}, \multimap)$ *a high-level implementation system.*

1) Normalisation*:* t *is* $\to_{\mathtt{f}}$*-normalising iff it is* $\multimap$*-normalising.*
2) Projection*: if* $d : t \multimap^* u$ *then* $d{\downarrow} : t \to_{\mathtt{f}}^* u{\downarrow}$,

Moreover, the overhead of $\multimap$ *is, depending on the system:*

1) locally bounded*: quadratic, i.e.* $|d| = O(|d{\downarrow\!\!\downarrow}|^2)$.
2) globally bounded*: linear, i.e.* $|d| = O(|d{\downarrow\!\!\downarrow}|)$.

For the low-level part, in contrast to [1], we rely on abstract machines, introduced in the next section.

Let us see our framework at work. We have the following result:

Theorem 4. $(\to_{\mathtt{f}}, \multimap_{\mathtt{f}})$ *is a high-level implementation system.*

Note the absence of complexity bounds. In fact, $(\to_{\mathtt{f}}, \multimap_{\mathtt{f}})$ is not even locally bounded. Let t^n here be defined by $t^1 = t$ and $t^{n+1} = t^n t$, and $u_n := (\lambda x.x^n)A$. Then $d : u_n \multimap_{\mathtt{m}}\multimap_{\mathtt{e}}^n A^n[x \leftarrow A]$ is a counter-example to local boundedness. Moreover, the Explicit FBC also suffers of size explosion, *i.e.* implementing a single step may take exponential time. In Sect. XI the introduction of useful sharing will solve these issues.

B. Low-Level Implementation: Abstract Machines

Introducing Distilleries: an abstract machine $\mathtt{M}$ is meant to implement a strategy $\multimap$ via a *distillation*, *i.e.* a decoding function $\underline{\cdot}$. A machine has a state s, given by a *code* $\overline{t}$, *i.e.* a l-term t without ES and not considered up to α-equivalence, and some data-structures like stacks, dumps, environments, and eventually heaps. The data-structures are used to implement the search of the next $\multimap$-redex and some form of parsimonious substitution, and they distill to evaluation contexts for $\multimap$. Every state s decodes to a term $\underline{s}$, having the shape $F\langle t\rangle$, where t is a l-term and F is some kind of evaluation context for $\multimap$.

A machine computes using transitions, whose union is noted $\rightsquigarrow$, of two types. The *principal* one, noted $\rightsquigarrow_{\mathtt{p}}$, corresponds to the firing of a rule defining $\multimap$. In doing so, the machine can differ from the calculus implemented by a transformation of the evaluation context to an equivalent one, up to a structural congruence $\equiv$. The *commutative* transitions, noted $\rightsquigarrow_{\mathtt{c}}$, implement the search for the next redex to be fired by rearranging the data-structures to single out a new evaluation context, and they are invisible on the calculus. The names reflect a proof-theoretical view, as machine transitions can be seen as cut-elimination steps [15], [28]. Garbage collection is here simply ignored, as in the LSC it can always be postponed.

To preserve correctness, structural congruence $\equiv$ is required to commute with evaluation $\multimap$, *i.e.* to satisfy

$$\left(\begin{array}{lll} t \multimap r & & t \multimap r \\ \equiv & \Rightarrow \exists q \text{ s.t. } & \equiv \quad \equiv \\ u & & u \multimap q \end{array}\right) \wedge \left(\begin{array}{lll} t & & t \multimap r \\ \equiv & \Rightarrow \exists r \text{ s.t. } & \equiv \quad \equiv \\ u \multimap q & & u \multimap q \end{array}\right)$$

for each of the rules of $\multimap$, preserving the kind of rule. In fact, this means that $\equiv$ is a *strong* bisimulation (*i.e. one* step to *one* step) with respect to $\multimap$. Strong bisimulations formalise transformations which are transparent with respect to the behaviour, even at the level of complexity, because they can be retarded without affecting the length of evaluation:

Lemma 1 ($\equiv$ Postponement)**.** *If $\equiv$ is a strong bisimulation and $t\ (\rightarrow \cup \equiv)^* u$ then $t \rightarrow^* \equiv u$ and the number and kind of steps of $\multimap$ in the two reduction sequences is the same.*

We can finally introduce distilleries, *i.e.* systems where a strategy $\multimap$ simulates a machine $\mathtt{M}$ up to structural equivalence $\equiv$ (via the decoding $\underline{\cdot}$).

Definition 2. *A* distillery $\mathtt{D} = (\mathtt{M}, \multimap, \equiv, \underline{\cdot})$ *is given by:*

1) *An* abstract machine $\mathtt{M}$*, given by*
   a) *a deterministic labeled* transition *system $\rightsquigarrow$ on states s;*
   b) *a distinguished class of states deemed* initial*, in bijection with closed l-terms and from which one obtains the* reachable *states by applying $\rightsquigarrow^*$;*
   c) *a partition of the labels of the transition system $\rightsquigarrow$ as:*
      - principal *transitions, noted* $\rightsquigarrow_{\mathtt{p}}$,
      - commutative *transitions, noted* $\rightsquigarrow_{\mathtt{c}}$*;*
2) *a deterministic* strategy $\multimap$*;*
3) *a* structural equivalence $\equiv$ *on terms s.t. it is a strong bisimulation with respect to $\multimap$;*
4) *a* distillation $\underline{\cdot}$*, i.e. a decoding function from states to terms, s.t. on reachable states:*
   - Principal*: $s \rightsquigarrow_{\mathtt{p}} s'$ implies $\underline{s} \multimap \equiv \underline{s'}$,*
   - Commutative*: $s \rightsquigarrow_{\mathtt{c}} s'$ implies $\underline{s} \equiv \underline{s'}$.*

We will soon prove that a distillery implies a simulation theorem, but we want a stronger form of relationship. Additional hypothesis are required to obtain the converse simulation, handle explicit substitution, and talk about complexity bounds.

Some terminology first. An *execution* ρ is a sequence of transition from an initial state. With $|\rho|$, $|\rho|_p$, and $|\rho|_c$ we denote respectively the length, the number of principal, and commutative transitions of ρ. The *size* of a term is noted $|t|$.

Definition 3 (Distillation Qualities)**.** *A distillery is*

- Reflective *when on reachable states:*
  - Termination*: $\rightsquigarrow_{\mathtt{c}}$ terminates;*
  - Progress*: if $\underline{s}$ reduces then s is not final.*
- Explicit *when*
  - Partition*: principal transitions are partitioned into* multiplicative $\rightsquigarrow_m$ *and* exponential $\rightsquigarrow_e$*, like for the strategy $\multimap$.*
  - Explicit decoding*: the partition is preserved by the decoding, i.e.*
    - Multiplicative*: $s \rightsquigarrow_m s'$ implies $\underline{s} \multimap_{\mathtt{m}} \equiv \underline{s'}$;*
    - Exponential*: $s \rightsquigarrow_e s'$ implies $\underline{s} \multimap_{\mathtt{e}} \equiv \underline{s'}$;*
- Bilinear *when it is reflective and*
  - Execution Length*: given an execution ρ from an initial term t, the number of commutative steps $|\rho|_c$ is linear in both $|t|$ and $|\rho|_p$ (with a slightly stronger dependency on $|t|$, due to the time needed to recognise a normal form), i.e. if $|\rho|_c = O((1 + |\rho|_p) \cdot |t|)$.*
  - Commutative*: $\rightsquigarrow_{\mathtt{c}}$ is implementable on RAM in a constant number of steps;*
  - Principal*: $\rightsquigarrow_{\mathtt{p}}$ is implementable on RAM in $O(|t|)$ steps.*

A reflective distillery is enough to obtain a bisimulation between the strategy $\multimap$ and the machine $\mathtt{M}$, that is strong up to structural equivalence $\equiv$. With $|\rho|_{\mathtt{m}}$ and $|\rho|_{\mathtt{e}}$ we denote respectively the number of multiplicative and exponential transitions of ρ.

Theorem 5 (Correctness and Completeness)**.** *Let $\mathtt{D}$ be a reflective distillery and s an initial state.*

1) Strong Simulation*: for every execution $\rho : s \rightsquigarrow^* s'$ there is a derivation $d : \underline{s} \multimap^* \equiv \underline{s'}$ s.t. $|\rho|_p = |d|$.*
2) Reverse Strong Simulation*: for every derivation $d : \underline{s} \multimap^* t$ there is an execution $\rho : s \rightsquigarrow^* s'$ s.t. $t \equiv \underline{s'}$ and $|\rho|_p = |d|$.*

Moreover, if $\mathtt{D}$ is explicit then $|\rho|_{\mathtt{m}} = |d|_{\mathtt{m}}$ and $|\rho|_{\mathtt{e}} = |d|_{\mathtt{e}}$.

Bilinearity, instead, is crucial for the low-level theorem.

Theorem 6 (Low-Level Implementation Theorem)**.** *Let $\multimap$ be a strategy on terms with ES s.t. there exists a bilinear distillery*

Table II
GLAM: DATA-STRUCTURES, DECODING AND TRANSITIONS

$$\begin{array}{rcl@{\qquad}rcl} \phi & ::= & \overline{t} \mid (\overline{t},\pi) & E,E' & ::= & \epsilon \mid [x{\leftarrow}\overline{t}] : E \\ \pi,\pi' & ::= & \epsilon \mid \phi : \pi & s,s' & ::= & (D,\overline{t},\pi,E) \\ D,D' & ::= & \epsilon \mid D : (\overline{t},\pi) & & & \end{array}$$

$$\begin{array}{rcl@{\qquad}rcl} \underline{\epsilon} & := & \langle\cdot\rangle & \underline{[x{\leftarrow}\overline{t}] : E} & := & \langle\langle\cdot\rangle[x{\leftarrow}\overline{t}]\rangle\underline{E} \\ \underline{\phi : \pi} & := & \langle\langle\cdot\rangle\underline{\phi}\rangle\underline{\pi} & \underline{F_s} & := & \langle\underline{D}\langle\underline{\pi}\rangle\rangle\underline{E} \\ \underline{(\overline{t},\pi)} & := & \langle t\rangle\underline{\pi} & \underline{s} & := & \underline{F_s}\langle t\rangle \\ \underline{D : (\overline{t},\pi)} & := & \underline{D}\langle\langle\overline{t}\langle\rangle\rangle\underline{\pi}\rangle & \text{where } s = (D,\overline{t},\pi,E) & & \end{array}$$

$$\begin{array}{ccc|c|ccc|c|c|c} D & \overline{t}\overline{u} & \pi & E & \rightsquigarrow_{\mathtt{c}_1} & D : (\overline{t},\pi) & \overline{u} & \epsilon & E \\ D & lx.\overline{t} & \overline{u} : \pi & E & \rightsquigarrow_{\mathtt{m}} & D & \overline{t} & \pi & [x{\leftarrow}\overline{u}]E \\ D : (\overline{t},\pi) & a & \pi' & E & \rightsquigarrow_{\mathtt{c}_3} & D & \overline{t} & (a,\pi') : \pi & E \\ D : (\overline{t},\pi) & lx.\overline{u} & \epsilon & E & \rightsquigarrow_{\mathtt{c}_2} & D & \overline{t} & lx.\overline{u} : \pi & E \\ D & x & \pi & E_1[x{\leftarrow}\overline{u}]E_2 & \rightsquigarrow_{\mathtt{e}} & D & \overline{u}^{\alpha} & \pi & E_1[x{\leftarrow}\overline{u}]E_2 \end{array}$$

where $\overline{u}^{\alpha}$ is any code α-equivalent to $\overline{u}$ that preserves well-naming of the machine, i.e. such that any bound name in $\overline{u}^{\alpha}$ is fresh with respect to those in D, π and $E_1[x{\leftarrow}\overline{u}]E_2$.

$\mathtt{D} = (\mathtt{M}, \multimap, \equiv, \underline{\cdot})$. *Then a $\multimap$-derivation d is implementable on RAM machines in $O((1+|d|)\cdot|t|)$ steps, i.e. bilinear in the size of the initial term t and the length of the derivation $|d|$.*

Proof: given $d : t \multimap^n u$ by Theorem 5.2 there is an execution $\rho : s \rightsquigarrow^* s'$ s.t. $u \equiv \underline{s'}$ and $|\rho|_p = |d|$. The number of RAM steps to implement ρ is the sum of the number for the commutative and the principal transitions. By bilinearity, $|\rho|_c = O((1+|\rho|_p)\cdot|t|)$ and so all the commutative transitions in ρ require $O((1+|\rho|_p)\cdot|t|)$ steps, because a single one takes a constant number of steps. Again by bilinearity, each principal one requires $O(|t|)$ steps, and so all the principal transitions together require $O(|\rho|_p \cdot |t|)$ steps. ∎

We will discuss three distilleries, summarised in Table IV (page 11), and two of them will be bilinear. The machines will be sophisticated, so that we will first present a machine for the inefficient Explicit FBC (Sect. VIII, called GLAM), that we will later refine with useful sharing (Sect. XII, GLAMoUr) and with renaming chains elimination (Sect. XIV, Unchaining GLAMoUr).

Let us point out an apparent discrepancy with the literature. For the simpler case without symbols, the number of commutative steps of the abstract machine studied in [3] is truly linear (and not bilinear), *i.e.* it does not dependent on the size of the initial term. Three remarks:

1) *Complete Evaluation*: it is true only for evaluation to normal form, while our low-level theorem is also valid for both any prefix of the evaluation and diverging evaluations.
2) *Normal Form Recognition*: it relies on the fact that closed normal forms (*i.e.* values) can be recognised in constant time, by simply checking the topmost constructor. With symbols checking if a term is normal requires time linear in its size, and so linearity is simply not possible.
3) *Asymptotically Irrelevant*: the dependency from the initial term disappears from the number of commutative transitions but still affects the cost of the principal ones, because every exponentials transition copies a subterm of the initial term, and thus it takes $O(|t|)$ time.

VIII. An Inefficient Distillery: the GLAM Machine

In this section we introduce the GLAM machine and show that it distills to the Explicit FBC. The distillery is inefficient, because Explicit FBC suffers of size explosion, but it is a good case study to present distilleries before the optimisations. Moreover, it allows to show an unexpected fact: while adding useful sharing to the calculus will be a quite tricky and technical affair (Sect. XI), adding usefulness to the GLAM will be surprisingly simple (Sect. XII), and yet tests of usefulness will only require constant time.

The machine of this section is the Global LAM (GLAM). The name is due to a similar machine, based on *local* environments, introduced in [15] and called LAM—standing for Leroy Abstract Machine. The GLAM differs from the LAM in two respects: 1) it uses *global* rather than local environments, and 2) it has an additional rule to handle constructors.

Data-Structures: at the machine level, *terms* are replaced by *codes*, *i.e.* terms not considered up to α-equivalence. To distinguish codes from terms, we over-line codes like in $\overline{t}$.

States (noted $s, s', \ldots$) of the abstract machine are made out of a *context dump* D, a *code* t, an *argument stack* π, and a global environment E, defined by the grammars in Table II. To save space, sometimes we write $[x{\leftarrow}\overline{t}]E$ for $[x{\leftarrow}\overline{t}] : E$. Note that stacks may contain pairs $(\overline{t},\pi)$ of a code and a stack, used to code the application of $\overline{t}$ to the stack π. We choose this representation to implement commutative rules in constant time.

The Machine: the machine transitions are given in Table II. Note that the multiplicative one $\rightsquigarrow_{\mathtt{m}}$ puts a new entry in the environment, while the exponential one $\rightsquigarrow_{\mathtt{e}}$ performs a clashing-avoiding substitution from the environment. The idea is that the principal transitions $\rightsquigarrow_{\mathtt{m}}$ and $\rightsquigarrow_{\mathtt{e}}$ implement $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ while the commutative transitions $\rightsquigarrow_{\mathtt{c}_1}$, $\rightsquigarrow_{\mathtt{c}_2}$, and $\rightsquigarrow_{\mathtt{c}_3}$ locate and expose the next redex following a right-to-left strategy.

The commutative rule $\rightsquigarrow_{\mathtt{c}_1}$ forces evaluation to be right-to-left on applications: the machine processes first the argument $\overline{u}$, saving the left sub term $\overline{t}$ on the dump together with its current stack π. The role of $\rightsquigarrow_{\mathtt{c}_2}$ and $\rightsquigarrow_{\mathtt{c}_3}$ is to backtrack to the saved sub-term. Indeed, when the argument, *i.e.* the current code, is finally put in normal form, encoded by a *stack item*

Table III
CONTEXT AND RELATIVE UNFOLDING

Context Unfolding

$$\begin{array}{lcl} \langle\cdot\rangle\downarrow & := & \langle\cdot\rangle \\ (tS)\downarrow & := & t\downarrow S\downarrow \\ (St)\downarrow & := & S\downarrow t\downarrow \\ S[x\leftarrow t]\downarrow & := & S\downarrow\{x\leftarrow t\downarrow\} \end{array}$$

Relative Unfolding

$$\begin{array}{lcl} t\downarrow_{\langle\cdot\rangle} & := & t\downarrow \\ t\downarrow_{uS} & := & t\downarrow_S \\ t\downarrow_{Su} & := & t\downarrow_S \\ t\downarrow_{S[x\leftarrow u]} & := & t\downarrow_S\{x\leftarrow u\downarrow\} \end{array}$$

Relative Context Unfolding

$$\begin{array}{lcl} S'\downarrow_{\langle\cdot\rangle} & := & S'\downarrow \\ S'\downarrow_{uS} & := & S'\downarrow_S \\ S'\downarrow_{Su} & := & S'\downarrow_S \\ S'\downarrow_{S[x\leftarrow u]} & := & S'\downarrow_S\{x\leftarrow u\downarrow\} \end{array}$$

ϕ, the stack item is pushed on the stack, and the machine backtracks to the pair on the dump.

The Distillery: machines start an execution on *initial states* defined as $(\epsilon, \overline{t}, \epsilon, \epsilon)$, *i.e.* obtained by taking the term, seen now as the code $\overline{t}$, and setting to ϵ the other machine components. A state represents a term—given by the code—and an evaluation context, that for the GLAM is obtained by decoding D, π, and E. The decoding $\underline{\cdot}$ (or distillation) function is defined in Table II. Note that stacks are decoded to contest in postfix notation for plugging. To improve readability, when we decode machines, we will denote $W\langle t\rangle$ with $\langle t\rangle W$, if the component occurs on the right of t in the machine representation.

A machine state is *closed* when all free variables in any component of the state are bound in E or, equivalently, when $\underline{s}$ is closed in the usual sense. It is *well-named* when all variables bound in the state are distinct. We require well-namedness as a machine invariant to allow every environment entry $[x\leftarrow\overline{t}]$ to be global (i.e. to bind x everywhere in the machine state). From now on, the initial state associated to a term t has as code the term obtained α-converting t to make it well-named.

For every machine we will have invariants, in order to prove the properties of a distillery. They are always proved by induction over the length of the execution, by a simple inspection of the transitions. For the GLAM:

Lemma 2 (GLAM Invariants)**.** *Let $s = (D, u, \pi, E)$ be a state reachable from an initial code $\overline{t}$. Then:*

1) Closure*: s is closed and well-named;*
2) Value*: values in components of s are sub-terms of $\overline{t}$;*
3) Fireball*: every term in π, in E, and in every stack in D is a fireball;*
4) Contextual Decoding*: $\underline{E}$, $\underline{D}$, $\underline{\pi}$, and F_s are evaluation contexts;*

The invariants are used to prove the following theorem.

Theorem 7 (GLAM Distillation)**.** *$(GLAM, \multimap_{\mathtt{f}}, \equiv, \underline{\cdot})$ is a reflective explicit distillery. In particular, let s be a reachable state reachable:*

1) Commutative*: if $s \rightsquigarrow_{\mathtt{c}_{1,2,3}} s'$ then $\underline{s} = \underline{s'}$;*
2) Multiplicative*: if $s \rightsquigarrow_{\mathtt{m}} s'$ then $\underline{s} \multimap_{\mathtt{m}} \equiv \underline{s'}$;*
3) Exponential*: if $s \rightsquigarrow_{\mathtt{e}} s'$ then $\underline{s} \multimap_{\mathtt{e}} \underline{s'}$.*

Since the Explicit FBC suffers of size-explosion, an exponential step (and thus an exponential transition) may duplicate a subterm that is exponentially bigger than the input. Then $(\text{GLAM}, \multimap_{\mathtt{f}}, \equiv, \underline{\cdot})$ does not satisfy bilinearity, for which every exponential transition has to have linear complexity in the size of the input.

IX. INTERLUDE: RELATIVE UNFOLDINGS

Now we define some notions for weak contexts that will be implicitly instantiated to all kind of contexts in the paper. In particular, we define substitution over contexts, and then use it to define the unfolding of a context, and the more general notion of relative unfolding.

Implicit substitution on weak contexts W is defined by

$$\begin{array}{lcl} \langle\cdot\rangle\{x\leftarrow u\} & := & \langle\cdot\rangle \\ (tW)\{x\leftarrow u\} & := & t\{x\leftarrow u\}W\{x\leftarrow u\} \\ (Wt)\{x\leftarrow u\} & := & W\{x\leftarrow u\}t\{x\leftarrow u\} \\ W[y\leftarrow t]\{x\leftarrow u\} & := & W\{x\leftarrow u\}[y\leftarrow t\{x\leftarrow u\}] \\ t[y\leftarrow W]\{x\leftarrow u\} & := & t\{x\leftarrow u\}[y\leftarrow W\{x\leftarrow u\}] \end{array}$$

Lemma 3. *Let t be a term and W a weak context. Then $W\langle t\rangle\{x\leftarrow u\} = W\{x\leftarrow u\}\langle t\{x\leftarrow u\}\rangle$.*

Now, we would like to extend the unfolding to contexts, but in order to do so we have to restrict the notion of context. Indeed, whenever the hole of a context is inside an ES, the unfolding may erase or duplicate the hole, producing a term or a multi-context, which we do not want. Thus, we turn to (weak) *shallow contexts*, defined by:

$$S, S', S'' ::= \langle\cdot\rangle \mid St \mid tS \mid S[x\leftarrow t].$$

(note the absence of the production $t[x\leftarrow S]$).

Now, we define in Table III *context unfolding $S\downarrow$, unfolding $t\downarrow_S$ of a term t relative to a shallow context S* and *unfolding $S'\downarrow_S$ of a shallow context S' relative to a shallow context S*.

Relative unfoldings have a number of properties, summed up in the appendix (page 24). Last, a definition that will be important in the next section.

Definition 4 (Applicative Context)**.** *A shallow context S is* applicative *when its hole is applied to a sub term u, i.e. if $S = S'\langle Lu\rangle$.*

X. INTRODUCING USEFUL SHARING

Beware: this and the next sections will heavily use contexts and notions about them as defined in Sect. VI and Sect. IX, in particular the notions of *shallow* context, *applicative context*, and *relative unfolding*.

Introducing Useful Reduction: note that the substitution steps in the size exploding family do not create redexes. We want to restrict the calculus so that these *useless* steps are avoided. The idea of useful sharing, is to trigger an exponential

redex only if it will somehow contribute to create a multiplicative redex. Essentially, one wants only the exponential steps

$$F\langle x\rangle[x\leftarrow L\langle f\rangle] \multimap_{\mathtt{e}} L\langle F\langle f\rangle[x\leftarrow f]\rangle$$

s.t. F is applicative and f is a value, so that the firing creates a multiplicative redex. Such a change of approach, however, has consequences on the whole design of the system. Indeed, since some substitutions are delayed, the present requirements for the rules might not be met. Consider:

$$(\lambda x.t)y[y\leftarrow ab]$$

we want to avoid substituting ab for the argument y, but we also want that evaluation does not stop, *i.e.* that $(\lambda x.t)y[y\leftarrow ab] \to_{\mathtt{m}} t[x\leftarrow y[y\leftarrow ab]]$. To accomodate such a dynamics, our definitions have to be *up to unfolding*, *i.e.* *fireballs* have to be replaced by *terms unfolding to fireballs*. There are 4 subtle things about useful reduction.

1) Multiplicatives and Variables. The idea is that the multiplicative rule becomes

$$L\langle lx.t\rangle L'\langle u\rangle \quad \mapsto_{\mathtt{m}} \quad L\langle t[x\leftarrow L'\langle u\rangle]\rangle$$

where it is the unfolding $L'\langle u\rangle{\downarrow}$ of the argument $L'\langle u\rangle$ that is a fireball, and not necessarily $L'\langle u\rangle$ itself. Note that sometimes variables are valid arguments of multiplicative redexes, and consequently substitutions may contain variables.

2) Exponentials and Future Creations. The exponential rule involves contexts, and is trickier to make it useful. A first approximation of useful exponential step is

$$F\langle x\rangle[x\leftarrow L\langle u\rangle] \quad \mapsto_{\mathtt{e}} \quad L\langle F\langle u\rangle[x\leftarrow u]\rangle$$

where $L\langle u\rangle{\downarrow}$ is a *value* (*i.e.* it is not a inert) and F is applicative, so that—after eventually many substitution steps, when x becomes $u{\downarrow}$—a multiplicative redex will pop out.

Note that an useful exponential step does not always *immediately* create a multiplicative redex. Consider the following step (where I is the identity):

$$(xI)[x\leftarrow y][y\leftarrow I] \multimap_{\mathtt{e}} (yI)[x\leftarrow y][y\leftarrow I] \tag{1}$$

No multiplicative redex has been created yet, but step (1) is useful because the *next* exponential step creates a multiplicative redex:

$$(yI)[x\leftarrow y][y\leftarrow I] \multimap_{\mathtt{e}} (II)[x\leftarrow y][y\leftarrow I]$$

3) Evaluation and Evaluable Contexts. The delaying of useless substitutions impacts also on the notion of evaluation context F, used in the exponential rule. For instance, the following exponential step should be useful

$$((xI)y)[x\leftarrow I][y\leftarrow ab] \multimap_{\mathtt{e}} ((II)y)[x\leftarrow I][y\leftarrow ab]$$

but the context $((\langle\cdot\rangle I)y)[x\leftarrow I][y\leftarrow ab]$ isolating x is not an evaluation context, it only unfolds to one. We then need a notion of evaluation context up to unfolding. The intuition is that a shallow context S is *evaluable* if $S{\downarrow}$ is an evaluation context (see Sect. IX for the definition of context unfolding), and it is *useful* if it is evaluable and applicative. The exponential rule then should rather be:

$$S\langle x\rangle[x\leftarrow L\langle u\rangle] \quad \mapsto_{\mathtt{e}} \quad L\langle S\langle u\rangle[x\leftarrow u]\rangle$$

where $u{\downarrow}$ is a *value* and S is *useful*.

4) Context Closure vs Global Rules. Such a definition, while close to the right one, still misses a fundamental point, *i.e.* the *global* nature of useful steps. Evaluation rules are indeed defined by a further *closure by contexts*, *i.e.* a step takes place in a certain shallow context S'. Of course, S' has to be evaluable, but there is more. Such a context, in fact, may also give an essential contribution to the usefulness of a step. Let us give an example. Consider the following exponential step

$$(xx)[x\leftarrow y] \multimap_{\mathtt{e}} (yx)[x\leftarrow y]$$

By itself it is not useful, since y is not a value nor unfolds to one. If we plug that redex in the context $S := \langle\cdot\rangle[y\leftarrow I]$, however, then y unfolds to a value in S, as $y{\downarrow}_S = y{\downarrow}_{\langle\cdot\rangle[y\leftarrow\lambda z.z]} = \lambda z.z$, and the step becomes:

$$(xx)[x\leftarrow y][y\leftarrow\lambda z.z] \multimap_{\mathtt{e}} (yx)[x\leftarrow y][y\leftarrow\lambda z.z] \tag{2}$$

No multiplicative redex has been created yet, but step (2) is useful because it is essential for the creation given by the *next* exponential step:

$$(yx)[x\leftarrow y][y\leftarrow\lambda z.z] \multimap_{\mathtt{e}} ((\lambda z.z)x)[x\leftarrow y][y\leftarrow\lambda z.z]$$

Note, indeed, that $(\lambda z.z)x$ gives a useful multiplicative redex, because x unfolds to a fireball in its context $\langle\cdot\rangle[x\leftarrow y][y\leftarrow\lambda z.z]$.

Summing up, the useful or useless character of a step depends crucially on the surrounding context. Therefore useful rules have to be *global*: rather than given as axioms closed by evaluable contexts, they will involve the surrounding context itself and impose conditions about it.

The Useful FBC, presented in the next section, formalises these ideas. We will prove it to be a locally bounded implementation of $\to_{\mathtt{f}}$, obtaining our fist high-level implementation theorem.

XI. The Useful Fireball Calculus

For the Useful FBC, terms, values, and substitution contexts are unchanged (with respect to the Explicit FBC), and we use *shallow contexts* S as defined in Sect. IX. An *initial term* is still a closed term with no explicit substitutions.

The new key notion is that of *evaluable* context.

Definition 5 (Evaluable and Useful Contexts)**.** Evaluable *(shallow) contexts are defined by the inference system in Table V. A context is* useful *if it is evaluable and applicative (being applicative is easily seen to be preserved by unfolding).*

Point 1 of the following Lemma 4 guarantees that evaluable contexts capture the intended semantics suggested in the previous section. Point 2 instead provides an equivalent inductive formulation that does not mention relative unfoldings. The definition in Table V can be thought has been *from the*

Table IV
DISTILLERIES IN THE PAPER + REWRITING RULES FOR THE USEFUL FBC

Calculus	*Machine*	RULE (ALREADY CLOSED BY CONTEXTS)	SIDE CONDITIONS
FBC $\to_{\mathtt{f}}$		$S\langle L\langle lx.t\rangle u\rangle \multimap_{\mathtt{um}} S\langle L\langle t[x\leftarrow u]\rangle\rangle$	$S\langle Lu\rangle$ is useful
Explicit FBC $\multimap_{\mathtt{f}}$	GLAM		
Useful FBC $\multimap_{\mathtt{uf}}$	GLAMoUr	$S\langle S'\langle x\rangle[x\leftarrow L\langle u\rangle]\rangle \multimap_{\mathtt{ue}} S\langle L\langle S'\langle u\rangle[x\leftarrow u]\rangle\rangle$	$S\langle S'[x\leftarrow L\langle u\rangle]\rangle$ is useful
Unchaining FBC $\multimap_{\mathtt{of}}$	Unchaining GLAMoUr		$u \neq u'[y\leftarrow w]$ and $u\downarrow_{S\langle L\rangle} = v$

outside, while the lemma give a characterisation *from the inside*, relating sub-terms to their surrounding sub-context.

Lemma 4.

1) *If S is evaluable then $S\downarrow$ is an evaluation context.*
2) *S is evaluable iff $u\downarrow_{S'}$ is a fireball whenever $S = S'\langle S''u\rangle$ or $S = S'\langle S''[x\leftarrow u]\rangle$.*

Rewriting Rules: the two rewriting rules $\multimap_{\mathtt{um}}$ and $\multimap_{\mathtt{ue}}$ are defined in Table IV, and we use $\multimap_{\mathtt{uf}}$ for $\multimap_{\mathtt{um}} \cup \multimap_{\mathtt{ue}}$. The rules are *global*, *i.e.* they do not factor as a rule followed by a contextual closure. As already explained, the context has to be taken into account, to understand if the step is useful to multiplicative redexes.

In rule $\multimap_{\mathtt{um}}$, the requirement that whole context around the abstraction is useful guarantees that the argument u unfolds to a fireball in its context. Note also that in $\multimap_{\mathtt{ue}}$ this is not enough, we have to be sure that such an unfolding is a value, otherwise it will not be useful to multiplicative redexes. Moreover, the rule requires $u \neq u'[y\leftarrow w]$, to avoid copying substitutions.

A detailed study of useful evaluation (in the appendix) shows that:

Theorem 8 (Quadratic High-Level Implementation)**.** *$(\to_{\mathtt{f}}, \multimap_{\mathtt{uf}})$ is a locally bounded high-level implementation system, and so it has a quadratic overhead wrt $\to_{\mathtt{f}}$.*

Moreover, the structural equivalence $\equiv$ is a strong bisimulation also with respect to $\multimap_{\mathtt{uf}}$.

Proposition 4 ($\equiv$ is a Strong Bisimulation wrt $\multimap_{\mathtt{uf}}$)**.** *Let $\mathtt{x} \in \{\mathtt{um}, \mathtt{ue}\}$. Then, $t \equiv u$ and $t \multimap_{\mathtt{x}} t'$ implies that there exists u' such that $u \multimap_{\mathtt{x}} u'$ and $t' \equiv u'$.*

XII. THE GLAMOUR MACHINE

Here we refine the GLAM with a very simple tagging of stacks and environments, in order to implement useful sharing. The idea is that every term in the stack or in the environment carries a label $l \in \{v, A\}$ indicating if it unfolds (relatively to the environment) to a value or to a inert.

The grammars are identical to the GLAM, up to labels:

$$\begin{array}{rcl} l & ::= & v \mid A \\ \pi, \pi' & ::= & \epsilon \mid \phi^l : \pi \end{array} \qquad\qquad \begin{array}{rcl} E, E' & ::= & \epsilon \mid [x\leftarrow\phi^l] : E \end{array}$$

The decoding of the various machine components is identical to that for the GLAM, up to labels that are ignored. The state context, however, now is noted S_s, as it is not necessary an evaluation context.

The transitions are in Table VI. They are obtained from those of the GLAM by:

1) *Backtracking instead of performing a useless substitution*: there are two new backtracking cases $\rightsquigarrow_{\mathtt{c}_4}$ and $\rightsquigarrow_{\mathtt{c}_5}$ (that in the GLAM were handled by the exponential transition), corresponding to avoided useless duplications: $\rightsquigarrow_{\mathtt{c}_4}$ backtracks when the entry ϕ to substitute is marked A (as it unfolds to a inert) and $\rightsquigarrow_{\mathtt{c}_5}$ backtracks when the term is marked v but the stack is empty (*i.e.* the context is not applicative).

Table V
EVALUABLE SHALLOW CONTEXTS

$$\frac{}{\langle\cdot\rangle \text{ is evaluable}} \qquad \frac{S \text{ is eval.} \qquad t\downarrow \text{ is a fireball}}{St \text{ is evaluable}}$$

$$\frac{S \text{ is evaluable}}{tS \text{ is evaluable}} \qquad \frac{S\{x\leftarrow t\downarrow\} \text{ is eval.} \qquad t\downarrow \text{ is a fireball}}{S[x\leftarrow t] \text{ is evaluable}}$$

Table VI
TRANSITIONS OF THE GLAMOUR

D	$\overline{t}\,\overline{u}$	π	E	$\rightsquigarrow_{\mathtt{c}_1}$	$D : (\overline{t}, \pi)$	$\overline{u}$	ϵ	E
D	$lx.\overline{t}$	$\phi^l : \pi$	E	$\rightsquigarrow_{\mathtt{um}}$	D	$\overline{t}$	π	$[x\leftarrow\phi^l]E$
$D : (\overline{t}, \pi)$	$lx.\overline{u}$	ϵ	E	$\rightsquigarrow_{\mathtt{c}_2}$	D	$\overline{t}$	$(lx.\overline{u})^v : \pi$	E
$D : (\overline{t}, \pi)$	a	π'	E	$\rightsquigarrow_{\mathtt{c}_3}$	D	$\overline{t}$	$(a, \pi')^A : \pi$	E
$D : (\overline{t}, \pi)$	x	π'	$E_1[x\leftarrow\phi^A]E_2$	$\rightsquigarrow_{\mathtt{c}_4}$	D	$\overline{t}$	$(x, \pi')^A : \pi$	$E_1[x\leftarrow\phi^A]E_2$
$D : (\overline{t}, \pi)$	x	ϵ	$E_1[x\leftarrow\overline{u}^v]E_2$	$\rightsquigarrow_{\mathtt{c}_5}$	D	$\overline{t}$	$x^v : \pi$	$E_1[x\leftarrow\overline{u}^v]E_2$
D	x	$\phi^l : \pi$	$E_1[x\leftarrow\overline{u}^v]E_2$	$\rightsquigarrow_{\mathtt{ue}}$	D	$\overline{u}^\alpha$	$\phi^l : \pi$	$E_1[x\leftarrow\overline{u}^v]E_2$

where $\overline{u}^\alpha$ is any code α-equivalent to $\overline{u}$ that preserves well-naming of the machine.

2) *Substituting only when it is useful*: the exponential transition is applied only when the term to substitute has label v and the stack is non-empty.

Lemma 5 (GLAMoUr Invariants)**.** *Let $s = (D, \overline{u}, \pi, E)$ be a state reachable from an initial code $\overline{t}$. Then:*

1) Closure*: s is closed and well named;*
2) Value*: values in components of s are sub-terms of $\overline{t}$;*
3) Fireball*: $\overline{t}{\downarrow_{\underline{E}}}$ is a fireball (of kind l) for every code $\overline{t}^{\,l}$ in π, E, and in every stack of D;*
4) Evaluability*: $\underline{E}$, $\underline{D}{\downarrow_{\underline{E}}}$, $\underline{\pi}{\downarrow_{\underline{E}}}$, and S_s are evaluable contexts;*
5) Environment Size*: the length of the global environment E is bound by $|\rho|_{\mathtt{m}}$.*

Theorem 9 (GLAMoUr Distillation)**.** *($GLAMoUr, \multimap_{\mathtt{uf}}, \equiv, \underline{\cdot}$) is a reflective explicit distillery. In particular, let s be a reachable state:*

1) Commutative*: if $s \rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}} s'$ then $\underline{s} = \underline{s'}$;*
2) Multiplicative*: if $s \rightsquigarrow_{\mathtt{um}} s'$ then $\underline{s} \multimap_{\mathtt{um}} \equiv \underline{s'}$;*
3) Exponential*: if $s \rightsquigarrow_{\mathtt{ue}} s'$ then $\underline{s} \multimap_{\mathtt{ue}} \underline{s'}$.*

In fact, the distillery is even bilinear, as we now show. The proof employs the following definition of size of a state.

Definition 6. *The* size *of codes and states is defined by:*

$$\begin{array}{rclrcl} |x| = |a| & := & 1 & |\overline{t}\overline{u}| & := & |\overline{t}| + |\overline{u}| + 1 \\ |lx.\overline{t}| & := & |\overline{t}| + 1 & |(D, \overline{t}, \pi, E)| & := & |\overline{t}| + \Sigma_{(\overline{u},\pi)\in D}|\overline{u}| \end{array}$$

Lemma 6 (Size Bounded)**.** *Let $s = (D, \overline{u}, \pi, E)$ be a state reached by an execution ρ of initial code $\overline{t}$. Then $|s| \leq (1 + |\rho|_{\mathtt{ue}})|\overline{t}| - |\rho|_c$.*

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 $\overline{t}\overline{u}$ 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* $\rightsquigarrow_{\mathtt{c}_{2,3,4,5}}$: these rules consume the current code, so they decrease the measure of at least 1. *Multiplicative*: it consumes the lambda abstraction. *Exponential*: it modifies the current code by replacing a variable (of size 1) with a value $\overline{v}$ coming from the environment. Because of Lemma 5.2, $\overline{v}$ is a sub-term of $\overline{t}$ and the dump size increment is bounded by $|\overline{t}|$. ∎

Corollary 1 (Bilinearity of $\rightsquigarrow_{\mathtt{c}}$)**.** *Let s be a state reached by an execution ρ of initial code $\overline{t}$. Then $|\rho|_c \leq (1 + |\rho|_{\mathtt{e}})|\overline{t}|$.*

Finally, we obtain our first implementation theorem.

Theorem 10 (Useful Implementation)**.**

1) Low-Level Bilinear Implementation*: a $\multimap_{\mathtt{uf}}$-derivation d is implementable on RAM in $O((1 + |d|) \cdot |t|)$ (i.e. bilinear) steps.*
2) Low + High Quadratic Implementation*: a $\rightarrow_{\mathtt{f}}$-derivation d is implementable on RAM in $O((1 + |d|^2) \cdot |t|)$ steps, i.e. linear in the size of the initial term t and quadratic in the length of the derivation $|d|$.*

XIII. The Unchaining FBC

In this section we start by analysing why the Useful FBC has a quadratic overhead. We then refine it, obtaining the Unchaining FBC, that we will prove to have only a linear overhead wrt the FBC. The optimisation has to do with the order in which chains of useful substitutions are performed.

Analysis of Useful Substitution Chains: in the Useful FBC, whenever there is a situation like

$$(x_1 A)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow x_n][x_n \leftarrow v]$$

the $\multimap_{\mathtt{uf}}$ strategy performs $n + 1$ exponential steps $\multimap_{\mathtt{ue}}$ replacing x_1 with x_2, then x_2 with x_3, and so on, until v is finally substituted on the head

$$\begin{array}{ll} (x_n A)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow x_n][x_n \leftarrow v] & \multimap_{\mathtt{ue}} \\ (vA)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow x_n][x_n \leftarrow v] & \end{array}$$

and a multiplicative redex can be fired. Any later occurrence of x_1 will trigger the same chain of exponential steps again. Because the length n of the chain is bounded by the number of previous multiplicative steps (local bound property), the overall complexity of the machine is quadratic in the number of multiplicative steps. In our previous work [16], we showed that to reduce the complexity to linear it is enough to perform substitution steps in reverse order, modifying the chains while traversing them. The idea is that in the previous example one should rather have a smart reduction $\multimap_{\mathtt{oe}}$ ($\mathtt{o}$ stays for optimised, as $\mathtt{u}$ is already used for useful reduction) following the chain of substitutions and performing:

$$\begin{array}{ll} (x_1 A)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow x_n][x_n \leftarrow v] & \multimap_{\mathtt{oe}} \\ (x_1 A)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow v][x_n \leftarrow v] & \multimap_{\mathtt{oe}} \\ \dots & \\ (x_1 A)[x_1 \leftarrow v] \dots [x_{n-1} \leftarrow v][x_n \leftarrow v] & \multimap_{\mathtt{oe}} \\ (vA)[x_1 \leftarrow v] \dots [x_{n-1} \leftarrow v][x_n \leftarrow v] & \end{array}$$

Later occurrences of x_1 will no longer trigger the chain, because it has been *unchained* by traversing it the first time.

Unfortunately, introducing such an optimisation for useful reduction is hard. In the shown example, that has a very simple form, it is quite easy to define what *following the chain* means. For the distillation machinery to work, however, we need our rewriting rules to be stable by structural equivalence, whose action is a rearrangement of substitutions through the term structure. Then the substitutions $[x_i \leftarrow x_{i+1}]$ of the example can be spread all over the term, interleaved by applications and other substitutions, and even nested one into the other (like in $[x_i \leftarrow x_{i+1}[x_{i+1} \leftarrow x_{i+2}]]$). This makes the specification of optimised useful reduction a quite technical affair.

Chain Contexts: reconsider a term like in the example, $(xA)[x_1 \leftarrow x_2][x_2 \leftarrow x_3][x_3 \leftarrow x_4][x_4 \leftarrow v]$. We want the next step to substitute on x_4 so we should give a status to the context $C := (xA)[x_1 \leftarrow x_2][x_2 \leftarrow x_3][x_3 \leftarrow \langle\cdot\rangle]$. The problem is that C can be deformed by structural equivalence $\equiv$ as

$$C' := (x[x_1 \leftarrow x_2[x_2 \leftarrow x_3]]A)[x_3 \leftarrow \langle\cdot\rangle]$$

and so this context has to be caught too. We specify these context in Table VII as *chain contexts* C, defined using the

Table VII
IDENTITY, CHAIN, AND CHAIN-STARTING CONTEXT + REWRITING RULES OF THE UNCHAINING FBC

$$\begin{array}{rcl} I, I' & ::= & \langle\cdot\rangle \mid I\langle x\rangle[x\leftarrow I'] \mid I[x\leftarrow t] \\ C, C' & ::= & S\langle x\rangle[x\leftarrow I] \mid C\langle x\rangle[x\leftarrow I] \mid S\langle C\rangle \end{array}$$

$$\begin{array}{rcl} \overleftarrow{S\langle y\rangle[y\leftarrow I]}^{x} & := & S[y\leftarrow I\langle x\rangle] \\ \overleftarrow{C\langle y\rangle[y\leftarrow I]}^{x} & := & \overleftarrow{C}^{y}[y\leftarrow I\langle x\rangle] \\ \overleftarrow{S\langle C\rangle}^{x} & := & S\langle\overleftarrow{C}^{x}\rangle \end{array}$$

RULE (ALREADY CLOSED BY CONTEXTS)	SIDE CONDITION
$S\langle L\langle lx.t\rangle u\rangle \multimap_{\mathtt{om}} S\langle L\langle t[x\leftarrow u]\rangle\rangle$	$S\langle\langle\cdot\rangle u\rangle$ is useful
$S\langle S'\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle \multimap_{\mathtt{oes}} S\langle L\langle S'\langle v\rangle[x\leftarrow v]\rangle\rangle$	$S\langle S'[x\leftarrow L\langle v\rangle]\rangle$ is useful
$S\langle C\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle \multimap_{\mathtt{oec}} S\langle L\langle C\langle v\rangle[x\leftarrow v]\rangle\rangle$	$S\langle\overleftarrow{C}^{x}[x\leftarrow L\langle v\rangle]\rangle$ is useful

auxiliary notion of *identity context* I, that captures a simpler form of chain (note that both notions are not shallow).

Given a chain context C, we will need to retrieve the point where the chain started, *i.e.* the shallow context isolating the variable at the left end of the chain (x_1 in the example). We are now going to define an operation associating to every chain context its *chain-starting (shallow) context*. To see the two as contexts of a same term, we need also to provide the sub-term that we will put in C (that will always be a variable). The chain-starting context $\overleftarrow{C}^{x}$ associated to the chain context C (with respect to x) is defined in Table VII.

For our example $C := (xA)[x_1\leftarrow x_2][x_2\leftarrow x_3][x_3\leftarrow\langle\cdot\rangle]$ we have $\overleftarrow{C}^{x_4} = (\langle\cdot\rangle A)[x_1\leftarrow x_2][x_2\leftarrow x_3][x_3\leftarrow x_4]$, as expected.

Rewriting Rules: the rules of the Unchaining FBC are in Table VII. Note that the exponential rule splits in two, the ordinary *shallow* case $\multimap_{\mathtt{oes}}$ (now constrained to values) and the *chain* case $\multimap_{\mathtt{oec}}$ (where the new definition play a role). They could be merged, but for the complexity analysis and the relationship with the next machine is better to distinguish them. We use $\multimap_{\mathtt{oe}}$ for $\multimap_{\mathtt{oes}} \cup \multimap_{\mathtt{oec}}$, and $\multimap_{\mathtt{of}}$ for $\multimap_{\mathtt{om}} \cup \multimap_{\mathtt{oe}}$. Note the use of $\overleftarrow{C}^{x}$ in the third side condition.

A. Linearity: Multiplicative vs Exponential Analysis

To prove that $\multimap_{\mathtt{of}}$ implements $\to_{\mathtt{f}}$ with a global bound, and thus with a linear overhead, we need to show that the global number of exponential steps ($\multimap_{\mathtt{oe}}$) in a $\multimap_{\mathtt{of}}$-derivation is bound by the number of multiplicative steps ($\multimap_{\mathtt{om}}$). We need the following invariant.

Lemma 7 (Subterm Invariant)**.** *Let t be a l-term and $d : t \multimap^{*} u$. Then every value in u is a value in t.*

A substitution $t[x\leftarrow u]$ is *basic* if u has the form $L\langle y\rangle$. The *basic size* $|t|_{\mathtt{b}}$ of t is the number of its basic substitutions.

Lemma 8 (Steps and Basic Size)**.**

1) *If $t \multimap_{\mathtt{oes}} u$ then $|u|_{\mathtt{b}} = |t|_{\mathtt{b}}$;*
2) *If $t \multimap_{\mathtt{oec}} u$ then $|t|_{\mathtt{b}} > 0$ and $|u|_{\mathtt{b}} = |t|_{\mathtt{b}} - 1$;*
3) *If $t \multimap_{\mathtt{om}} u$ then $|u|_{\mathtt{b}} = |t|_{\mathtt{b}}$ or $|u|_{\mathtt{b}} = |t|_{\mathtt{b}} + 1$.*

Lemma 9. *Let t be initial and $d : t \multimap^{*}_{\mathtt{of}} u$. Then $|u|_{\mathtt{b}} \leq |d|_{\mathtt{om}} - |d|_{\mathtt{oec}}$.*

Proof: by induction on $|d|$. If $|d| = 0$ the statement holds. If $|d| > 0$ consider the last step $w \multimap_{\mathtt{of}} u$ of d and the prefix $e : t \multimap^{*}_{\mathtt{of}} w$ of d. By *i.h.*, $|w|_{\mathtt{b}} \leq |e|_{\mathtt{om}} - |e|_{\mathtt{oec}}$. Cases of $w \multimap_{\mathtt{of}} u$.

Shallow Exponential Step $\multimap_{\mathtt{oes}}$:

$$\begin{array}{lll} |u|_{\mathtt{b}} & \leq_{L.8.1} & |w|_{\mathtt{b}} - 1 \\ & \leq_{\textit{i.h.}} & |e|_{\mathtt{om}} - |e|_{\mathtt{oec}} - 1 \\ & = & |e|_{\mathtt{om}} - (|e|_{\mathtt{oec}} + 1) \quad = \quad |d|_{\mathtt{om}} - |d|_{\mathtt{oec}} \end{array}$$

Chain Exponential Step $\multimap_{\mathtt{oec}}$:

$$|u|_{\mathtt{b}} =_{L.8.2} |w|_{\mathtt{b}} \leq_{\textit{i.h.}} |e|_{\mathtt{om}} - |e|_{\mathtt{oec}} = |d|_{\mathtt{om}} - |d|_{\mathtt{oec}}$$

Multiplicative Step $\multimap_{\mathtt{om}}$:

$$\begin{array}{lll} |u|_{\mathtt{b}} & \leq_{L.8.3} & |w|_{\mathtt{b}} + 1 \\ & \leq_{\textit{i.h.}} & |e|_{\mathtt{om}} - |e|_{\mathtt{oec}} + 1 \\ & = & e + 1 - |e|_{\mathtt{oec}} \quad = \quad |d|_{\mathtt{om}} - |d|_{\mathtt{oec}} \end{array}$$ ■

Corollary 2 (Linear Bound on Chain Exponential Steps)**.** *Let t be initial and $d : t \multimap^{*}_{\mathtt{of}} u$. Then $|d|_{\mathtt{oec}} \leq |d|_{\mathtt{om}}$.*

Next, we bound shallow steps.

Lemma 10 (Linear Bound on Shallow Exponential Steps)**.** *Let t be initial and $d : t \multimap^{*}_{\mathtt{of}} u$. Then $|d|_{\mathtt{oes}} \leq |d|_{\mathtt{om}}$.*

Proof: first note that if $t \multimap_{\mathtt{oes}} u$ then $u \multimap_{\mathtt{om}} w$, because by definition $\multimap_{\mathtt{oes}}$ can fire only if it creates a $\multimap_{\mathtt{om}}$-redex. Such a fact and determinism of $\multimap_{\mathtt{of}}$ together imply $|d|_{\mathtt{oes}} \leq |d|_{\mathtt{om}} + 1$, because every $\multimap_{\mathtt{oes}}$ step is matched by the eventual $\multimap_{\mathtt{om}}$ steps that follows it immediately. However, note that in t there are no explicit substitutions so that the first step is necessarily an unmatched $\multimap_{\mathtt{om}}$ step. Thus $|d|_{\mathtt{oes}} \leq |d|_{\mathtt{om}}$. ■

Theorem 11 (Linear Bound on Exponential Steps)**.** *Let t be initial and $d : t \multimap^{*}_{\mathtt{of}} u$. Then $|d|_{\mathtt{oe}} \leq 2 \cdot |d|_{\mathtt{om}}$.*

Proof: by definition, $|d|_{\mathtt{oe}} = |d|_{\mathtt{oec}} + |d|_{\mathtt{oes}}$. By Corollary 2, $|d|_{\mathtt{oec}} \leq |d|_{\mathtt{om}}$ and by Lemma 10 $|d|_{\mathtt{oes}} \leq |d|_{\mathtt{om}}$, and so $|d|_{\mathtt{oe}} \leq 2 \cdot |d|_{\mathtt{om}}$. ■

We presented the interesting bit of the proof of our improved high-level implementation theorem, which follows. The remaining details are in the appendix.

Theorem 12 (Linear High-Level Implementation)**.** *$(\to_{\mathtt{f}}, \multimap_{\mathtt{of}})$ is a globally bounded high-level implementation system, and so it has a linear overhead wrt $\to_{\mathtt{f}}$.*

Last, the structural equivalence $\equiv$ is a strong bisimulation also for the Unchaining FBC.

Proposition 5 ($\equiv$ is a Strong Bisimulation)**.** *Let $\mathtt{x} \in \{\mathtt{om}, \mathtt{oms}, \mathtt{omc}\}$. Then, $t \equiv u$ and $t \multimap_{\mathtt{x}} t'$ implies that there exists u' such that $u \multimap_{\mathtt{x}} u'$ and $t' \equiv u'$.*

Table VIII
TRANSITIONS OF THE UNCHAINING GLAMOUR

D	ϵ	$\overline{t}\overline{u}$	π	E	$\rightsquigarrow_{\mathtt{c}_1}$	$D : (\overline{t}, \pi)$	ϵ	$\overline{u}$	ϵ	E
D	ϵ	$lx.\overline{t}$	$\phi^l : \pi$	E	$\rightsquigarrow_{\mathtt{om}}$	D	ϵ	$\overline{t}$	π	$[x \leftarrow \phi^l]E$
$D : (\overline{t}, \pi)$	ϵ	$lx.\overline{u}$	ϵ	E	$\rightsquigarrow_{\mathtt{c}_2}$	D	ϵ	$\overline{t}$	$(lx.\overline{u})^v : \pi$	E
$D : (\overline{t}, \pi)$	ϵ	a	π'	E	$\rightsquigarrow_{\mathtt{c}_3}$	D	ϵ	$\overline{t}$	$(a, \pi')^A : \pi$	E
$D : (\overline{t}, \pi)$	ϵ	x	π'	$E_1[x \leftarrow \phi^A]E_2$	$\rightsquigarrow_{\mathtt{c}_4}$	D	ϵ	$\overline{t}$	$(x, \pi')^A : \pi$	$E_1[x \leftarrow \phi^A]E_2$
$D : (\overline{t}, \pi)$	ϵ	x	ϵ	$E_1[x \leftarrow \overline{u}^v]E_2$	$\rightsquigarrow_{\mathtt{c}_5}$	D	ϵ	$\overline{t}$	$x^v : \pi$	$E_1[x \leftarrow \overline{u}^v]E_2$
D	ϵ	x	$\phi^l : \pi$	$E_1[x \leftarrow \overline{v}^v]E_2$	$\rightsquigarrow_{\mathtt{oes}}$	D	ϵ	$\overline{v}^\alpha$	$\phi^l : \pi$	$E_1[x \leftarrow \overline{v}^v]E_2$
D	H	x	$\phi^l : \pi$	$E_1[x \leftarrow y^v]E_2$	$\rightsquigarrow_{\mathtt{c}_6}$	D	$H : x$	y	$\phi^l : \pi$	$E_1[x \leftarrow y^v]E_2$
D	$H : y$	x	$\phi^l : \pi$	$E^\bullet$	$\rightsquigarrow_{\mathtt{oec}}$	D	H	y	$\phi^l : \pi$	E°

with $E^\bullet := E_1[x \leftarrow \overline{v}^v]E_2[y \leftarrow x^v]E_3$, $E^\circ := E_1[x \leftarrow \overline{v}^v]E_2[y \leftarrow \overline{v}^{\alpha\, v}]E_3$, and where $\overline{v}^\alpha$ is any code α-equivalent to $\overline{v}$ that preserves well-naming of the machine.

XIV. UNCHAINING GLAMOUR

The Unchaining GLAMoUr machine, in Table VIII, behaves like the GLAMoUr machine until the code is a variable x_1 that is hereditarily bound in the global environment to a value via the chain $[x_1 \leftarrow x_2]^v \dots [x_n \leftarrow v]^v$. At this point the machine needs to traverse the chain until it finds the final binding $[x_n \leftarrow v]^v$, and then traverse again the chain in the opposite direction replacing every $[x_i \leftarrow x_{i+1}]^v$ entry with $[x_i \leftarrow v]^v$.

The forward traversal of the chain is implemented by a new commutative rule $\rightsquigarrow_{\mathtt{c}_6}$ that pushes the variables encountered in the chain on a new machine component, called the *chain heap*. The backward traversal is driven by the next variable popped from the heap, and it is implemented by a new exponential rule (the *chain* exponential rule, corresponding to that of the calculus). Most of the analyses performed on the GLAMoUr machine carry over to the Unchaining GLAMoUr without modifications.

Every old grammar is as before, and heaps are simply lists of variables, *i.e.* they are defined by $H ::= \epsilon \mid H : x$.

Decoding and Invariants: because of chain heaps and chain contexts, the decoding is involved.

First of all, note that there is a correlation between the chain and the environment, as the variables of a chain heap $H = x_1 : \dots : x_n$ need to have corresponding entries $[x_i \leftarrow x_{i+1}^v]$. More precisely, we will show that the following notion of compatibility is an invariant of the machine.

Definition 7 (Compatibility Heap-Environment)**.** *Let E be an environment and $H = x_1 : \dots : x_n$ be a heap. We say that H is* compatible *with E if either H is empty or $[x_i \leftarrow x_{i+1}^v] \in E$ for $i < n$, $[x_n \leftarrow x^v] \in E$, and $[x \leftarrow \phi^v] \in E$ for some ϕ^v.*

Given a state $s = (D, H, \overline{t}, \pi, E)$, the dump, the stack and the environment provide a shallow context $S_s := \langle \underline{D}\langle \underline{\pi} \rangle \rangle \underline{E}$ that will be shown to be evaluable, as for the GLAMoUr.

If the chain heap H is not empty, the current code $\overline{t}$ is somewhere in the middle of a chain inside the environment, and it is not apt to fill the state context S_s. The right code is the variable x_1 starting the chain heap $H = x_1 : \dots : x_n$, *i.e.*:

$$\overline{t}^{\epsilon} \quad := \quad \overline{t} \qquad\qquad \overline{t}^{x_1 : \dots : x_n} \quad := \quad x_1$$

Finally, a state decodes to a term as follows: $\underline{s} := S_s\langle \overline{t}^H \rangle$.

Lemma 11 (Unchaining GLAMoUr Invariants)**.** *Let $s = (D, H, \overline{u}, \pi, E)$ be a state reachable from an initial code $\overline{t}$.*

1) Closure*: s is closed and s is well named;*
2) Value*: values in components of s are sub-terms of $\overline{t}$;*
3) Fireball*: $\overline{t}\downarrow_{\underline{E}}$ is a fireball (of kind l) for every code $\overline{t}^l$ in π and E;*
4) Evaluability*: $\underline{E}$, $\underline{D}\downarrow_{\underline{E}}$, $\underline{\pi}\downarrow_{\underline{E}}$, and S_s are evaluable cont.;*
5) Environment Size*: the length of the global environment E is bound by $|\rho|_{\mathtt{m}}$.*
6) Compatible Heap*: if $H \neq \epsilon$ then the stack is not empty, $\overline{u} = x$, and H is compatible with E.*

We need additional decodings to retrieve the chain-starting context C in the side-condition of $\multimap_{\mathtt{oec}}$ rule, that—unsurprisingly—is given by the chain heap. Let $s = (D, H : y, \overline{t}, \pi, E)$ be a state s.t. $H : y$ is compatible with E. Note that compatibility gives $E = E_1[y \leftarrow \overline{t}^v]E_2$. Define the chain context C_s and the substitution context L_s as:

$$C_s \quad := \quad \langle \underline{D}\langle \langle y^H \rangle \underline{\pi} \rangle \rangle \underline{E_1}[y \leftarrow \langle \cdot \rangle] \qquad\qquad L_s \quad := \quad \underline{E_2}$$

The first point of the following lemma guarantees that C_s and L_s are well defined. The second point proves that filling $L_s\langle C_s \rangle$ with the current term gives exactly the decoding of the state $\underline{s} = S_s\langle y^H \rangle$, and that moreover the chain starts exactly on the evaluable context given by the state, *i.e.* that $S_s = L_s\langle \overleftarrow{C_s}^x \rangle$.

Lemma 12 (Heaps and Contexts)**.** *Let $s = (D, H : y, x, \pi, E)$ be a state s.t. $H : y$ is compatible with E. Then:*

1) *L_s is a substitution context and C_s is a chain context*
2) *s. t. $\underline{s} = S_s\langle y^H \rangle = L_s\langle C_s\langle x \rangle \rangle$ with $S_s = L_s\langle \overleftarrow{C_s}^x \rangle$*

We can now sum up.

Theorem 13 (Unchaining GLAMoUr Distillation)**.** *$(\mathit{Unchaining\ GLAMoUr}, \multimap_{\mathtt{of}}, \equiv, \underline{\cdot})$ is a reflective explicit distillery. In particular, let s be a reachable state:*

1) Commutative*: if $s \rightsquigarrow_{\mathtt{c}_{1,2,3,4,5,6}} s'$ then $\underline{s} = \underline{s'}$;*
2) Multiplicative*: if $s \rightsquigarrow_{\mathtt{om}} s'$ then $\underline{s} \multimap_{\mathtt{om}} \equiv \underline{s'}$;*
3) Shallow Exponential*: if $s \rightsquigarrow_{\mathtt{oes}} s'$ then $\underline{s} \multimap_{\mathtt{oes}} \underline{s'}$;*
4) Chain Exponential*: if $s \rightsquigarrow_{\mathtt{oec}} s'$ then $\underline{s} \multimap_{\mathtt{oec}} \underline{s'}$.*

A. Bilinearity: Principal vs Commutative Analysis

Bilinearity wrt $\rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}}$ is identical to that of the GLAMoUr, thus we omit it and focus on $\rightsquigarrow_{\mathtt{c}_6}$.

The size $|H|$ of a chain heap is its length as a list.

Lemma 13 (Linearity of $\rightsquigarrow_{\mathtt{c}_6}$)**.** *Let $s = (D, H, \overline{t}, \pi, E)$ be a state reached by an execution ρ. Then*

1) $|\rho|_{c_6} = |H| + |\rho|_{\mathtt{oec}}$.
2) $|H| \leq |\rho|_{\mathtt{m}}$.
3) $|\rho|_{c_6} \leq |\rho|_{\mathtt{m}} + |\rho|_{\mathtt{oec}} = O(|\rho|_p)$.

Proof: 1) By induction over $|\rho|$ and analysis of the last machine transition. The $\rightsquigarrow_{\mathtt{c}_6}$ steps increment the size of the heap. The $\rightsquigarrow_{\mathtt{oec}}$ steps decrement it. All other steps do not change the heap. *2)* By the compatible heap invariant (Lemma 11.6), $|H| \leq |E|$. By the environment size invariant (Lemma 11.5), $|E| \leq |\rho|_{\mathtt{m}}$. Then $|H| \leq |\rho|_{\mathtt{m}}$. *3)* Plugging Point 2 into Point 1. ∎

Corollary 3 (Bilinearity of $\rightsquigarrow_{\mathtt{c}}$)**.** *Let s be a state reached by an execution ρ of initial code $\overline{t}$. Then $|\rho|_c \leq (1 + |\rho|_{\mathtt{e}})|\overline{t}| + |\rho|_{\mathtt{m}} + |\rho|_{\mathtt{oec}} = O((1 + |\rho|_p) \cdot |\overline{t}|)$.*

Finally, we obtain the main result of the paper.

Theorem 14 (Useful Implementation)**.**

1) Low-Level Bilinear Implementation*: a $-\circ_{\mathtt{of}}$-derivation d is implementable on RAM in $O((1 + |d|) \cdot |t|)$ steps.*
2) Low + High Bilinear Implementation*: a $\rightarrow_{\mathtt{f}}$-derivation d is implementable on RAM in $O((1 + |d|) \cdot |t|)$ steps.*

Let us conclude with a remark. For our results to hold, the output of the computation has to be given in compact form, *i.e.* with ES. The unfolding a term t with ES may have size exponential in the size of t. It is important to show, then, that the common operations on λ-terms, and in particular equality checking (up to α-conversion), can be implemented efficiently on the shared representation, avoiding unfolding. In other words, we want to prove that ES are *succinct data structures*, in the sense of Jacobson [36].

Despite quadratic and quasi-linear recent algorithms [6], [37] for testing equality of terms with ES, we discovered that a linear algorithm can be obtained slightly modifying an algorithm already known quite some time before (1976!): the Paterson-Wegman linear unification algorithm [38] (better explained in [39]). The algorithm works on first order terms represented as DAGs, and unification boils down to equality checking when no metavariable occurs in the involved terms.

To apply the Paterson-Wegmar algorithm, we need to overcome two difficulties. The first one is that ES implement sharing explicitly: to represent the term tt sharing the two occurrences of t we need to introduce a variable and an ES, obtaining $xx[x \leftarrow t]$. On the contrary, the input to Paterson-Wegmar should be a DAG where the application node points directly twice to the root of t. The required change in representation can be easily computed in linear time in the size of the input. The second difficulty is that Paterson-Wegmar works on first-order terms, while we want to consider α-conversion. If we assume that occurrences of λ-bound variables point to their binder, two variables are α-equivalent when they point to nodes that have already been determined to be candidates for equality. The details of the adaptation of Paterson-Wegmar are left to a forthcoming publication.

Acknowledgements

A special acknowledgement to Ugo Dal Lago, to whom we owe the intuition that using labels may lead to a local and efficient implementation of useful sharing. We are also grateful to François Pottier, whose comments on a draft helped to improve the terminology and the presentation.

References

[1] B. Accattoli and U. Dal Lago, "Beta Reduction is Invariant, Indeed," in *CSL-LICS 2014*, 2014, p. 8. [Online]. Available: http://doi.acm.org/10.1145/2603088.2603105

[2] G. E. Blelloch and J. Greiner, "Parallelism in sequential functional languages," in *FPCA*, 1995, pp. 226–237.

[3] 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*, 2002, pp. 60–84. [Online]. Available: http://dx.doi.org/10.1007/3-540-36377-7_4

[4] U. Dal Lago and S. Martini, "The weak lambda calculus as a reasonable machine," *Theor. Comput. Sci.*, vol. 398, no. 1-3, pp. 32–50, 2008.

[5] ——, "On constructor rewrite systems and the lambda calculus," *Logical Methods in Computer Science*, vol. 8, no. 3, 2012. [Online]. Available: http://dx.doi.org/10.2168/LMCS-8(3:12)2012

[6] B. Accattoli and U. Dal Lago, "On the invariance of the unitary cost model for head reduction," in *RTA*, 2012, pp. 22–37.

[7] B. Accattoli, E. Bonelli, D. Kesner, and C. Lombardi, "A nonstandard standardization theorem," in *POPL*, 2014, pp. 659–670.

[8] B. Grégoire and X. Leroy, "A compiled implementation of strong reduction," in *(ICFP '02)*, 2002, pp. 235–246. [Online]. Available: http://doi.acm.org/10.1145/581478.581501

[9] L. Paolini and S. Ronchi Della Rocca, "Call-by-value solvability," *ITA*, vol. 33, no. 6, pp. 507–534, 1999.

[10] B. Accattoli and L. Paolini, "Call-by-value solvability, revisited," in *FLOPS*, 2012, pp. 4–16.

[11] A. Carraro and G. Guerrieri, "A semantical and operational account of call-by-value solvability," in *FOSSACS 2014*, 2014, pp. 103–118. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-54830-7_7

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

[13] R. Milner, M. Tofte, R. Harper, and D. Macqueen, *The Definition of Standard ML - Revised*. The MIT Press, May 1997. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262631

[14] D. Clément, T. Despeyroux, G. Kahn, and J. Despeyroux, "A simple applicative language: Mini-ml," in *LFP '86*. New York, NY, USA: ACM, 1986, pp. 13–27. [Online]. Available: http://doi.acm.org/10.1145/319838.319847

[15] B. Accattoli, P. Barenbaum, and D. Mazza, "Distilling abstract machines," in *ICFP 2014*, 2014, pp. 363–376. [Online]. Available: http://doi.acm.org/10.1145/2628136.2628154

[16] B. Accattoli and C. Sacerdoti Coen, "On the value of variables," in *WoLLIC 2014*, 2014, pp. 36–50. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44145-9_3

[17] R. Milner, "Local bigraphs and confluence: Two conjectures," *Electr. Notes Theor. Comput. Sci.*, vol. 175, no. 3, pp. 65–73, 2007.

[18] D. Kesner and S. Ó. Conchúir, "Milner's lambda calculus with partial substitutions," Paris 7 University, Tech. Rep., 2008.

[19] B. Accattoli and D. Kesner, "The structural λ-calculus," in *CSL*, 2010, pp. 381–395.

[20] N. G. de Bruijn, "Generalizing Automath by Means of a Lambda-Typed Lambda Calculus," in *Mathematical Logic and Theoretical Computer Science*, ser. Lecture Notes in Pure and Applied Mathematics, no. 106. Marcel Dekker, 1987, pp. 71–92.

[21] R. P. Nederpelt, "The fine-structure of lambda calculus," Eindhoven Univ. of Technology, Tech. Rep. CSN 92/07, 1992.

[22] B. Accattoli, "An abstract factorization theorem for explicit substitutions," in *RTA*, 2012, pp. 6–21.

[23] P. Curien, "An abstract framework for environment machines," *Theor. Comput. Sci.*, vol. 82, no. 2, pp. 389–402, 1991. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(91)90230-Y

[24] T. Hardin and L. Maranget, "Functional runtime systems within the lambda-sigma calculus," *J. Funct. Program.*, vol. 8, no. 2, pp. 131–176, 1998.
[25] M. Biernacka and O. Danvy, "A concrete framework for environment machines," *ACM Trans. Comput. Log.*, vol. 9, no. 1, 2007.
[26] F. Lang, "Explaining the lazy Krivine machine using explicit substitution and addresses," *Higher-Order and Symbolic Computation*, vol. 20, no. 3, pp. 257–270, 2007.
[27] P. Crégut, "Strongly reducing variants of the Krivine abstract machine," *Higher-Order and Symbolic Computation*, vol. 20, no. 3, pp. 209–230, 2007.
[28] Z. M. Ariola, A. Bohannon, and A. Sabry, "Sequent calculi and abstract machines," *ACM Trans. Program. Lang. Syst.*, vol. 31, no. 4, 2009.
[29] M. Fernández and N. Siafakas, "New developments in environment machines," *Electr. Notes Theor. Comput. Sci.*, vol. 237, pp. 57–73, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2009.03.035
[30] O. Danvy and I. Zerny, "A synthetic operational account of call-by-need evaluation," in *PPDP*, 2013, pp. 97–108.
[31] V. Danos and L. Regnier, "Head linear reduction," Tech. Rep., 2004.
[32] O. Danvy and L. R. Nielsen, "Refocusing in reduction semantics," BRICS, Tech. Rep. RS-04-26, 2004.
[33] M. Wand, "On the correctness of the krivine machine," *Higher-Order and Symbolic Computation*, vol. 20, no. 3, pp. 231–235, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10990-007-9019-8
[34] D. P. Friedman, A. Ghuloum, J. G. Siek, and O. L. Winebarger, "Improving the lazy krivine machine," *Higher-Order and Symbolic Computation*, vol. 20, no. 3, pp. 271–293, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10990-007-9014-0
[35] P. Sestoft, "Deriving a lazy abstract machine," *J. Funct. Program.*, vol. 7, no. 3, pp. 231–264, 1997. [Online]. Available: http://journals.cambridge.org/action/displayAbstract?aid=44087
[36] G. J. Jacobson, "Succinct static data structures," Ph.D. dissertation, Pittsburgh, PA, USA, 1988, aAI8918056.
[37] C. Grabmayer and J. Rochel, "Maximal sharing in the lambda calculus with letrec," in *ICFP 2014*, 2014, pp. 67–80. [Online]. Available: http://doi.acm.org/10.1145/2628136.2628148
[38] M. S. Paterson and M. N. Wegman, "Linear unification," in *STOC '76*. New York, NY, USA: ACM, 1976, pp. 181–186. [Online]. Available: http://doi.acm.org/10.1145/800113.803646
[39] D. de Champeaux, "About the Paterson-Wegman linear unification algorithm," *J. Comput. Syst. Sci.*, vol. 32, no. 1, pp. 79–90, Feb. 1986. [Online]. Available: http://dx.doi.org/10.1016/0022-0000(86)90003-6

APPENDIX A
PROOFS OMITTED FROM SECT. IV
(THE FIREBALL CALCULUS)

The following lemmas are required to prove Theorem 1.

Lemma 14. *Let t be a closed $\to_{\mathtt{f}}$-normal term. Then t is a fireball.*

Proof: by induction on t. Cases:

1) *Variable*. Impossible, because t is closed.
2) *Symbol* and *Abstraction*. Then t is a fireball.
3) *Application*. Then $t = uw$, with u and w both closed and $\to_{\mathtt{f}}$-normal. By *i.h.* they are both fireballs. Moreover, u cannot be a value, otherwise t would not be $\to_{\mathtt{f}}$-normal. Then it is a inert and t is a fireball. ■

Lemma 15. *Let t be a inert or a fireball. Then t is $\to_{\mathtt{f}}$-normal.*

Proof: by induction on t. If t is a value v or a symbol a then it is $\to_{\mathtt{f}}$-normal. Otherwise $t = Af$ and by *i.h.* both A and f are $\to_{\mathtt{f}}$-normal. Since A cannot be an abstraction, t is $\to_{\mathtt{f}}$-normal. ■

Lemma 16 (Determinism of $\to_{\mathtt{f}}$)**.** *Let t be a term. Then t has at most one $\to_{\mathtt{f}}$ redex.*

Proof: by induction on t. Cases:

1) *Variable*, *Symbol*, or *Abstraction*. No redexes.
2) *Application* $t = uw$. By *i.h.*, there are two cases for w:
   a) *w has exactly one $\to_{\mathtt{f}}$ redex*. Then t has a $\to_{\mathtt{f}}$ redex, because $u\langle\cdot\rangle$ is an evaluation context. Moreover, no $\to_{\mathtt{f}}$ redex for t can lie in u, because by Lemma 15 w is not a fireball, and so $\langle\cdot\rangle w$ is not an evaluation context.
   b) *w has no $\to_{\mathtt{f}}$ redexes*. If w is not a fireball then t has no redexes, because $\langle\cdot\rangle w$ is not an evaluation context. If w is a fireball we look at u. By *i.h.*, there are two cases:
      i) *u has exactly one $\to_{\mathtt{f}}$ redex*. Then t has a $\to_{\mathtt{f}}$ redex, because $\langle\cdot\rangle w$ is an evaluation context and w is a fireball. Uniqueness comes from the fact that w has no $\to_{\mathtt{f}}$ redexes.
      ii) *u has no $\to_{\mathtt{f}}$ redexes*. If u is not a fireball (and thus not a value) then t has no redexes. If u is a fireball there are two cases:
         - *u is a inert A*. Then t is a fireball.
         - *u is a value $lx.r$*. Then $t = (lx.r)w$ is a $\to_{\mathtt{f}}$ redex, because w is a fireball. Moreover, there are no other $\to_{\mathtt{f}}$ redexes, because evaluation does not go under abstractions and w is a fireball. ■

Proof of Theorem 1 (page 4)
Proof: by Lemma 16 and Lemma 14. ■

The following easy properties of substitution will be needed later.

Lemma 17.

1) Substitutions Commute*:* $t\{x\leftarrow u\}\{y\leftarrow w\} = t\{y\leftarrow w\}\{x\leftarrow u\{y\leftarrow w\}\}$*;*
2) Fireballs are Stable by Substitution*:*
   a) *If u is a inert then $u\{x\leftarrow t\}$ is a inert, and*
   b) *if u is a fireball then $u\{x\leftarrow t\}$ is a fireball.*
3) $\rightarrow_{\mathtt{f}}$ and Substitution Commute*: if $F\langle t\rangle \rightarrow_{\mathtt{f}} F\langle u\rangle$ with $t \mapsto_{\mathtt{f}} u$ then $F\langle t\rangle\{x\leftarrow w\} \rightarrow_{\mathtt{f}} F\langle u\rangle\{x\leftarrow w\}$ with $t\{x\leftarrow w\} \mapsto_{\mathtt{f}} u\{x\leftarrow w\}$.*

Proof:

1) By induction on t.
2) By induction on u.
   a) u is a inert. Cases:
      i) If $u = a$ then $u\{x\leftarrow t\} = a\{x\leftarrow t\} = a$ is a inert.
      ii) If $u = L'\langle A\rangle L''\langle f\rangle$ then by *i.h.* $L'\langle A\rangle\{x\leftarrow t\}$ is a inert and $L''\langle f\rangle\{x\leftarrow t\}$ is a fireball, and so $u\{x\leftarrow t\} = (L'\langle A\rangle L''\langle f\rangle)\{x\leftarrow t\} = L'\langle A\rangle\{x\leftarrow t\}L''\langle f\rangle\{x\leftarrow t\}$ is a inert.
   b) *u is a fireball.* Cases:
      i) *u is a value* $\lambda x.w$. Then $u\{x\leftarrow t\} = \lambda x.w\{x\leftarrow t\}$, which is a value, *i.e.* a fireball.
      ii) *u is a inert* A. Then by Point 2a $u\{x\leftarrow t\}$ is a inert, *i.e.* a fireball.
3) By induction on F. Cases:
   a) *Empty context* $F = \langle\cdot\rangle$. If $t = (\lambda y.r)f \mapsto_{\mathtt{f}} r\{y\leftarrow f\} = u$ then

$$\begin{array}{lll} t\{x\leftarrow w\} & = & \\ ((\lambda y.r)f)\{x\leftarrow w\} & = & \text{(def. of } \cdot\{\cdot\leftarrow\cdot\}) \\ (\lambda y.r\{x\leftarrow w\})f\{x\leftarrow w\} & \rightarrow_{\mathtt{f}} & \\ r\{x\leftarrow w\}\{y\leftarrow f\{x\leftarrow w\}\} & = & \text{(Point 1)} \\ r\{y\leftarrow f\}\{x\leftarrow w\} & = & \\ u\{x\leftarrow w\} & & \end{array}$$

   b) *Application Right* $F = rF'$. Then

$$\begin{array}{lll} F\langle t\rangle\{x\leftarrow w\} & = & \\ (rF'\langle t\rangle)\{x\leftarrow w\} & = & \\ r\{x\leftarrow w\}F'\langle t\rangle\{x\leftarrow w\} & \rightarrow_{\mathtt{f}} & \textit{(i.h.)} \\ r\{x\leftarrow w\}F'\langle u\rangle\{x\leftarrow w\} & = & \\ (rF'\langle u\rangle)\{x\leftarrow w\} & = & \\ F\langle u\rangle\{x\leftarrow w\} & & \end{array}$$

   c) *Application Left* $F = F'f$. Then

$$\begin{array}{lll} F\langle t\rangle\{x\leftarrow w\} & = & \\ (F'\langle t\rangle f)\{x\leftarrow w\} & = & \\ F'\langle t\rangle\{x\leftarrow w\}f\{x\leftarrow w\} & \rightarrow_{\mathtt{f}} & \textit{(i.h.} \text{ \& Point 2b)} \\ F'\langle u\rangle\{x\leftarrow w\}f\{x\leftarrow w\} & = & \\ (F'\langle u\rangle f)\{x\leftarrow w\} & = & \\ F\langle u\rangle\{x\leftarrow w\} & & \end{array}$$

■

APPENDIX B
PROOFS OMITTED FROM SECT. VI
(FIREBALLS AND EXPLICIT SUBSTITUTIONS)

A. Closed Normal Forms and Determinism

The first step is to identify the reduction invariants, the most important one being the shape of terms—called *proper*—that are produced by the strategy $\multimap_{\mathtt{f}}$ starting with a term without ES.

Definition 8 (Proper Term)**.** *A term t is* proper *if*

1) ES*: any explicit substitution in t contains an answer, and*
2) Value*: any value in t does not contain ES.*

We also say that an ES is proper when it contains a proper answer.

Note that initial terms (having no ES) are proper and so the next lemma applies in particular when the starting term is initial.

Lemma 18 (Proper Invariant)**.** *Let t be a proper term. If $t \multimap_{\mathtt{f}}^* u$ then u is proper.*

Proof: by induction on the length k of the derivation $t \multimap_{\mathtt{f}}^* u$. If $k = 0$ the statement is just the hypothesis. Otherwise $t \multimap_{\mathtt{f}}^{k-1} w \multimap_{\mathtt{f}} u$ and by *i.h.* w is proper. Note that 1) multiplicative steps create proper ES, and 2) exponential steps copy proper fireballs only out of values and ES, preserving properness. ■

We now characterize normal forms: the next three lemmas conclude that normal terms are answers, and that answers are fireballs up to unfolding.

Point 2.a of the next statement is given with respect to unfolding relative with shallow context (defined in Sect. IX, page 9) because it will be used in this more general form in later sections.

Lemma 19 (Properties of Answers)**.** *Let $t = L\langle u\rangle$. Answers are $\multimap_{\mathtt{f}}$-normal, do not decompose as $F\langle x\rangle$, and (relatively) unfold to fireballs. More precisely,*

1) *If u is a inert or a fireball then t is $\multimap_{\mathtt{f}}$-normal and it does not decompose as $F\langle x\rangle$,*
2) *Moreover,*
   a) *If u is a inert then $t{\downarrow_S}$ is a inert.*
   b) *If u is a fireball then $t{\downarrow_S}$ is a fireball.*

Proof: by induction on L. Cases:

- *Empty List* $\langle\cdot\rangle$. By induction on u.
  1) u is a inert. Cases:
     a) *u is a symbol* a. Then it is normal and clearly does not decompose as $F\langle x\rangle$. Moreover, $t{\downarrow_S} = a{\downarrow_S} = a$ is a inert.
     b) *u is a inert* $L'\langle A\rangle L''\langle f\rangle$. Then by *i.h.* both $L'\langle A\rangle$ and $L''\langle f\rangle$ are normal. Since A cannot be an abstraction, the topmost application cannot be a $\multimap_{\mathtt{m}}$-redex, and so u is normal.

For each of $L'\langle A\rangle$ and $L''\langle f\rangle$ *i.h.* gives that it does not decompose as $F\langle x\rangle$. Then u does not decompose either. Moreover, by *i.h.* $L'\langle A\rangle{\downarrow_S}$ is a inert and $L''\langle f\rangle{\downarrow_S}$ is a fireball, and so $t{\downarrow_S} =_{L.30.1} L'\langle A\rangle{\downarrow_S}L''\langle f\rangle{\downarrow_S}$ is a inert.

2) u *is a fireball*. Cases:
   a) u *is a value* $\lambda x.w$. Then it is normal and does not decompose as $F\langle x\rangle$. Moreover, $t{\downarrow} =_{L.30.1} \lambda x.w{\downarrow}$, which is a value, *i.e.* a fireball.
   b) u *is a inert* A. Given by the *i.h.*.

- *Non-Empty List* $L = L'[x{\leftarrow}w]$. By *i.h.*, $L'\langle u\rangle$ is normal and cannot be decomposed as $F\langle x\rangle$, and so there cannot be $\multimap_{\mathtt{e}}$ redexes involving $[x{\leftarrow}w]$. Thus t is normal.
  For the absence of a decomposition, note that—apart from the trivial decomposition $\langle L\langle u\rangle\rangle$, that is not of the form $F\langle x\rangle$—every decomposition of $L\langle u\rangle$ is obtained from a decomposition of $L'\langle u\rangle$ by appending $[x{\leftarrow}w]$, and so $L\langle u\rangle$ does not decompose as $F\langle x\rangle$.
  For the *moreover* part, by *i.h.* $L'\langle u\rangle$ verifies the statement for no matter which shallow context. Then $t{\downarrow_S} = L'\langle u\rangle[x{\leftarrow}w]{\downarrow_S} =_{L.30.6} L'\langle u\rangle{\downarrow_{S\langle\langle\cdot\rangle[x\leftarrow w]\rangle}}$ also verifies the statement. ■

Lemma 20 (Normal Form Characterization)**.** *Let t be a $\multimap_{\mathtt{f}}$-normal term.*

1) *Either t is an answer,*
2) *or $t = F\langle x\rangle$.*

Proof: by induction on t. Cases:

1) *Variable* $t = x$. Here Point 2 holds, while evidently Point 1 is false.
2) *Symbol* $t = a$. Here Point 1 holds, and Point 2 is false.
3) *Abstraction* $t = \lambda x.u$. Here Point 1 holds, and Point 2 is false.
4) *Application* $t = uw$. By *i.h.* we are in one of the following two cases for the right sub-term w:
   a) *Point 1 holds but not Point 2*. By Lemma 19, w is normal. Note that $\langle\cdot\rangle w$ is an evaluation context. The *i.h.* gives one of the following two cases for the left sub-term u:
      i) *Point 1 holds but not Point 2*. Given that both u and w do not satisfy Point 2, neither does t. Being an answer, w has the form $L\langle f\rangle$. Two cases:
         A) f *is a inert* A. Then $t = L\langle A\rangle w$ is the application of a inert to an answer, which is a inert—*i.e.* an answer—and Point 1 holds.
         B) f *is a value*. Then t is a $\multimap_{\mathtt{m}}$ redex, absurd.
      ii) *Point 2 holds but not Point 1*. Then Point 2 holds for t, because $\langle\cdot\rangle w$ is an evaluation context. Since u is not a inert, t is not answer, and Point 1 does not hold.
   b) *Point 2 holds but not Point 1*. Then Point 2 holds for t, because $u\langle\cdot\rangle$ is an evaluation context, and Point 1 does not, because w is not an answer.
5) *Substitution* $t = u[x{\leftarrow}w]$. Since $\langle\cdot\rangle[x{\leftarrow}w]$ is an evaluation context we can apply the *i.h.*, and fall into one of the two following cases:
   a) *Point 1 holds but not Point 2*. Then t is an answer, *i.e.* Point 1 holds. Note that since any non-empty evaluation context for t comes from an evaluation context for u, Point 2 holds for t iff it holds for u, *i.e.* it does not.
   b) *Point 2 holds but not Point 1*. Then $u = F'\langle y\rangle$ and we conclude taking $F := F'[x{\leftarrow}w]$. Note that it may be that $x = y$, but in that case w is not an answer (otherwise there would be a redex). There is no contradiction, because we are not assuming t to be proper (case in which one necessarily has $x \neq y$). ■

Corollary 4. *Let t be a closed proper $\multimap_{\mathtt{f}}$-normal term. Then t is an answer and $t{\downarrow}$ is $\to_{\mathtt{f}}$-normal.*

Proof: if t is $\multimap_{\mathtt{f}}$-normal then by Lemma 20 either t is an answer or it has the form $F\langle x\rangle$. Suppose that it has the form $F\langle x\rangle$. Since t is closed, F has a substitution on x, and since t is proper, that substitution contains an answer. Then t has a $\multimap_{\mathtt{e}}$-redex, absurd. Then t is an answer. By Lemma 19, $t{\downarrow}$ is a fireball. By Lemma 15, $t{\downarrow}$ is $\to_{\mathtt{f}}$-normal. ■

In order to prove determinism of the calculus, we need the notion of *position* and a final property of answers.

The position of a multiplicative redex is the context F in which the rule takes place, and this is standard. The position of an exponential redex $F'\langle F\langle x\rangle[x{\leftarrow}L\langle f\rangle]\rangle \multimap_{\mathtt{e}} S'\langle L\langle F\langle f\rangle[x{\leftarrow}f]\rangle\rangle$ is the context around the substituted variable, *i.e.* $F'\langle F[x{\leftarrow}L\langle f\rangle]\rangle$.

Given a term t, a redex is *contained* in a sub-term u if the whole rewriting pattern is contained in u. An exponential redex is *partially contained* in u if u contains the substituted variable (and then the position of the redex) but not the acting substitution.

Lemma 21 (Answers do not (Partially) Contain Redexes)**.** *Let $t = F\langle u\rangle$ be a term with u an answer. Then no redex of t can have its position in u.*

Proof: by Lemma 19, u is $\multimap_{\mathtt{f}}$-normal and so no $\multimap_{\mathtt{m}}$-redex of t can have its position in u. Moreover, u is not of the form $F\langle y\rangle$ and so no $\multimap_{\mathtt{e}}$-redex of t can be entirely nor partially contained in u. ■

Lemma 22 (Determinism)**.** *Let t be a term and F_1 and F_2 the positions of two redexes in t. Then $F_1 = F_2$.*

Proof: let $t = F_1\langle u\rangle$. By induction on F_1. Cases:

1) *Empty* $F_1 = \langle\cdot\rangle$. Cases:
   a) *Multiplicative Redex, i.e.* $t = L\langle\lambda x.r\rangle q$ with q an answer. By Lemma 21, F_2 cannot lie in $L\langle\lambda x.r\rangle$ nor in q. Then necessarily $F_2 = F_1 = \langle\cdot\rangle$.
   b) *Exponential Redex*. This case is impossible because the position of an exponential redex is the context around the substituted variable and if $F_1 = \langle\cdot\rangle$ then $t = x$ and there is no substitution acting on x.

2) *Right Application* $F_1 = rF_1'$ and $t = rF_1'\langle u\rangle$. By Lemma 21, $F_1'\langle u\rangle$ is not an answer and so F_2 does not lie in r, nor F_2 can be empty (*i.e.* $t = rF_1'\langle u\rangle$ cannot be a $\multimap_{\mathtt{m}}$-redex). Then, $F_2 = uF_2'$, and the statement follows from the *i.h.* applied to F_1' and F_2'.
3) *Left Application* $F_1 = F_1'L\langle f\rangle$ and and $t = F_1'\langle u\rangle L\langle f\rangle$. By Lemma 21, F_2 does not lie in $L\langle f\rangle$. And F_2 cannot be empty (*i.e.* the position of a $\multimap_{\mathtt{m}}$-redex), because then $F_1'\langle u\rangle$ would have the form $L\langle\lambda x.p\rangle$, *i.e.* it would be an answer, and so by Lemma 21 no redex positions can lie in $F_1'\langle u\rangle$, against the hypothesis of the case. Then, $F_2 = F_2'w$, and the statement follows from the *i.h.* applied to F_1' and F_2'.
4) *Substitution* $F_1 = F_1'[x\leftarrow w]$. Then necessarily $F_2 = F_2'[x\leftarrow w]$ (remember the position of a $\multimap_{\mathtt{e}}$-redex is given by the context around the substituted variable, and not by the one around the acting substitution) and the statement follows from the *i.h.*. ■

Corollary 5. *Let t be a proper closed term. Then either t contains exactly one $\multimap_{\mathtt{f}}$-redex, or t is an answer.*

Proof: by Lemma 22, t contains at most one redex. If it contains no redexes, then by Corollary 4 it is an answer. ■

B. Structural Equivalence

The aim is to prove the strong bisimulation of structural equivalence, whose proof relies on the next lemma.

Lemma 23. *The equivalence relation $\equiv$ preserves the "shapes" of $L\langle f\rangle$ and $F\langle x\rangle$. Formally:*

1) *If $L\langle f\rangle \equiv t$, then t is of the form $L'\langle g\rangle$.*
2) *If $F\langle x\rangle \equiv t$, with x not bound by F, then t is of the form $F'\langle x\rangle$, with x not bound by F'.*

Proof:

1) By induction on L.
2) By induction on F.

■

Now, we are ready for the bisimulation property.

Proof of Proposition 2 (page 6)

Proof:

Let $\Leftrightarrow$ be the symmetric closure of the union of the axioms defining $\equiv$, *i.e.* of $\equiv_{com} \cup \equiv_{@l} \cup \equiv_{@r} \cup \equiv_{[\cdot]}$. Note that $\equiv$ is the reflexive–transitive closure of $\Leftrightarrow$. The proof is in two parts:

(I) Prove the property holds for $\Leftrightarrow$, *i.e.* if $t \multimap_a u$ and $t \Leftrightarrow w$, there exists r s.t. $w \multimap_a r$ and $u \equiv r$.

(II) Prove the property holds for $\equiv$ (*i.e.* for many steps of $\Leftrightarrow$) by resorting to (I).

The proof of (II) is immediate by induction on the number of $\Leftrightarrow$ steps. The proof of (I) goes by induction on the rewriting step $\multimap$ (that, since $\multimap$ is closed by evaluation contexts, becomes a proof by induction on the evaluation context F). In principle, we should always consider the two directions of $\Leftrightarrow$. Most of the time, however, one direction is obtained by simply reading the diagram of the other direction bottom-up, instead than top-down; these cases are simply omitted, we distinguish the two directions only when it is relevant.

The proof of the strong bisimulation property is by induction on $\multimap$.

1) **Base case 1: multiplicative root step** $t = L\langle lx.t'\rangle L'\langle f\rangle \mapsto_{\mathtt{m}} u = L\langle t'[x\leftarrow L'\langle f\rangle]\rangle$**.** The nontrivial cases are when the $\Leftrightarrow$ step overlaps the pattern of the $\mathtt{m}$-redex. Note that by Lemma 23.1, if the $\Leftrightarrow$ is internal to $L'\langle f\rangle$, the proof is direct, since the $\mathtt{m}$-redex is preserved. More precisely, if $L'\langle f\rangle \Leftrightarrow L''\langle g\rangle$, we have:

$$\begin{array}{ccc} L\langle lx.t'\rangle\, L'\langle f\rangle & \multimap_{\mathtt{m}} & L\langle t'[x\leftarrow L'\langle f\rangle]\rangle \\ \Leftrightarrow & & \Leftrightarrow \\ L\langle lx.t'\rangle\, L''\langle g\rangle & \multimap_{\mathtt{m}} & L\langle t'[x\leftarrow L''\langle g\rangle]\rangle \end{array}$$

Consider the remaining possibilities for $\Leftrightarrow$:

a) *Commutation of independent substitutions* $\equiv_{com}$. The commutation of substitutions must be in L, *i.e.* L must be of the form $L_1\langle L_2[y\leftarrow u'][z\leftarrow w']\rangle$ with $z \notin \mathtt{fv}(u')$. Let $\widehat{L} := L_1\langle L_2[z\leftarrow w'][y\leftarrow u']\rangle$. Then:

$$\begin{array}{ccc} L\langle lx.t'\rangle\, L'\langle f\rangle & \multimap_{\mathtt{m}} & L\langle t'[x\leftarrow L'\langle f\rangle]\rangle \\ \equiv_{com} & & \equiv_{com} \\ \widehat{L}\langle lx.t'\rangle\, L'\langle f\rangle & \multimap_{\mathtt{m}} & \widehat{L}\langle t'[x\leftarrow L'\langle f\rangle]\rangle \end{array}$$

b) *Commutation with the left of an application* $\equiv_{@l}$. The diagram is:

$$\begin{array}{ccc} L\langle lx.t'\rangle[y\leftarrow q]\, L'\langle f\rangle & \multimap_{\mathtt{m}} & L\langle t'[x\leftarrow L'\langle f\rangle]\rangle[y\leftarrow q] \\ \equiv_{@} & & = \\ (L\langle lx.t'\rangle\, L'\langle f\rangle)[y\leftarrow q] & \multimap_{\mathtt{m}} & L\langle t'[x\leftarrow L'\langle f\rangle]\rangle[y\leftarrow q] \end{array}$$

c) *Commutation with the right of an application* $\equiv_{@r}$. The diagram is:

$$\begin{array}{ccc} L\langle lx.t'\rangle\, L'\langle f\rangle[y\leftarrow q] & \multimap_{\mathtt{e}} & L\langle t'[x\leftarrow L'\langle f\rangle[y\leftarrow q]]\rangle \\ & & \equiv_{[\cdot]} \\ \equiv_{@l} & & L\langle t'[x\leftarrow L'\langle f\rangle][y\leftarrow q]\rangle \\ & & \equiv^*_{com} \\ (L\langle lx.t'\rangle\, L'\langle f\rangle)[y\leftarrow q] & \multimap_{\mathtt{e}} & L\langle t'[x\leftarrow L'\langle f\rangle]\rangle[y\leftarrow q] \end{array}$$

d) *Composition of substitutions* $\equiv_{[\cdot]}$. The composition of substitutions must be in L, *i.e.* L must be of the form $L_1\langle L_2[y\leftarrow u'][z\leftarrow w']\rangle$ with $z \notin \mathtt{fv}(L_2\langle lx.t'\rangle)$. Let $\widehat{L} := L_1\langle L_2[y\leftarrow u'[z\leftarrow w']]\rangle$. Then:

$$\begin{array}{ccc} L\langle lx.t'\rangle\, L'\langle f\rangle & \multimap_{\mathtt{m}} & L\langle t'[x\leftarrow L'\langle f\rangle]\rangle \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ \widehat{L}\langle lx.t'\rangle\, L'\langle f\rangle & \multimap_{\mathtt{m}} & \widehat{L}\langle t'[x\leftarrow L'\langle f\rangle]\rangle \end{array}$$

2) **Base case 2: exponential root step** $t = F\langle x\rangle[x\leftarrow L\langle f\rangle] \mapsto_{\mathtt{e}} u = L\langle F\langle f\rangle[x\leftarrow f]\rangle$**.** Consider

first the case when the $\Leftrightarrow$-redex is internal to $F\langle x\rangle$. By Lemma 23.2 we know $\Leftrightarrow$ preserves the shape of $F\langle x\rangle$, *i.e.* $F\langle x\rangle \Leftrightarrow \widehat{F}\langle x\rangle$. Then:

$$\begin{array}{ccc} F\langle x\rangle[x{\leftarrow}L\langle f\rangle] & \xrightarrow{\quad\circ\quad} & L\langle F\langle f\rangle[x{\leftarrow}f]\rangle \\ \Leftrightarrow & & \equiv \\ \widehat{F}\langle x\rangle[x{\leftarrow}L\langle f\rangle] & \dashrightarrow & L\langle \widehat{F}\langle f\rangle[x{\leftarrow}f]\rangle \end{array}$$

If the $\Leftrightarrow$-redex is internal to one of the substitutions in L, the proof is similarly straightforward. Note that the $\Leftrightarrow$-redex has always a substitution at the root. The remaining possibilities are that such a substitution is in L and it interact with constructors outside L, or that it is precisely $[x{\leftarrow}L\langle f\rangle]$. Axiom by axiom:

a) *Commutation of independent substitutions* $\equiv_{com}$. The case where both commuted substitutions belong to L has already been treated. The remaining possibility is that $F = F'[y{\leftarrow}t']$ and $[x{\leftarrow}L\langle f\rangle]$ commutes with $[y{\leftarrow}t']$ (which implies $x \notin \mathtt{fv}(t')$). Then:

$$\begin{array}{ccc} F'\langle x\rangle[y{\leftarrow}t'][x{\leftarrow}L\langle f\rangle] & \xrightarrow{\quad\circ\quad} & L\langle F'\langle f\rangle[y{\leftarrow}t'][x{\leftarrow}f]\rangle \\ \equiv_{com} & & \equiv^*_{com} \\ F'\langle x\rangle[x{\leftarrow}L\langle f\rangle][y{\leftarrow}t'] & \dashrightarrow & L\langle F'\langle f\rangle[x{\leftarrow}f]\rangle[y{\leftarrow}t'] \end{array}$$

b) *Commutation with the left of an application* $\equiv_{@l}$. The only possibility is that the substitution $[x{\leftarrow}L\langle f\rangle]$ is commuted with the outermost application in $F\langle x\rangle$, *i.e.* $F = F'L'\langle g\rangle$. Then,

$$\begin{array}{ccc} (F'\langle x\rangle\, L'\langle g\rangle)[x{\leftarrow}L\langle f\rangle] & \xrightarrow{\quad\circ\quad} & L\langle (F'\langle f\rangle\, L'\langle g\rangle)[x{\leftarrow}f]\rangle \\ & & \equiv_{@l} \\ \equiv_{@l} & & L\langle F'\langle f\rangle[x{\leftarrow}f]L'\langle g\rangle\rangle \\ & & \equiv^*_{@l} \\ F'\langle x\rangle[x{\leftarrow}L\langle f\rangle]L'\langle g\rangle & \dashrightarrow & L\langle F'\langle f\rangle[x{\leftarrow}f]\rangle L'\langle g\rangle \end{array}$$

The $\equiv^*_{@l}$ step is justified by the fact that in the source term $(F'\langle x\rangle\, L'\langle g\rangle)[x{\leftarrow}L\langle f\rangle]$ the context L is only around f, and so it cannot capture variables in $L'\langle g\rangle$.

c) *Commutation with the right of an application* $\equiv_{@r}$. similarly to the previous case

$$\begin{array}{ccc} (t'\, F'\langle x\rangle)[x{\leftarrow}L\langle f\rangle] & \xrightarrow{\quad\circ\quad} & L\langle (t'\, F'\langle f\rangle)[x{\leftarrow}f]\rangle \\ & & \equiv_{@r} \\ \equiv_{@r} & & L\langle t'F'\langle f\rangle[x{\leftarrow}f]\rangle \\ & & \equiv^*_{@r} \\ t'F'\langle x\rangle[x{\leftarrow}L\langle f\rangle] & \dashrightarrow & t'L\langle F'\langle f\rangle[x{\leftarrow}f]\rangle \end{array}$$

The $\equiv^*_{@r}$ step is justified by the fact that in the source term $(F'\langle x\rangle\, L'\langle g\rangle)[x{\leftarrow}L\langle f\rangle]$ the context L is only around f, and so it cannot capture variables in t'.

d) *Composition of substitutions* $\equiv_{[\cdot]}$. The only possible case is that $[x{\leftarrow}L\langle f\rangle]$ is the outermost substitution composed by $\equiv_{[\cdot]}$. This is not possible if the rule is applied from left to right, since it would imply that $F\langle x\rangle = F'\langle x\rangle[y{\leftarrow}t']$ with $x \notin F'\langle x\rangle$, which is a contradiction.
Finally, if the $\equiv_{[\cdot]}$ rule is applied from right to left, L is of the form $L'[y{\leftarrow}t']$ and:

$$\begin{array}{ccc} F\langle x\rangle[x{\leftarrow}L'\langle f\rangle[y{\leftarrow}t']] & \xrightarrow{\quad\circ\quad} & L'\langle F\langle f\rangle[x{\leftarrow}f]\rangle[y{\leftarrow}t'] \\ \equiv_{[\cdot]} & & = \\ F\langle x\rangle[x{\leftarrow}L'\langle f\rangle][y{\leftarrow}t'] & \dashrightarrow & L'\langle F\langle x\rangle[x{\leftarrow}f]\rangle[y{\leftarrow}t'] \end{array}$$

3) **Inductive case 1: left of an application** $F = F'L\langle f\rangle$**.** The situation is:

$$t = F'\langle t'\rangle\, L\langle f\rangle \multimap w\, L\langle f\rangle = u$$

for some w. If the $\Leftrightarrow$ step is internal to $F'\langle t'\rangle$, the result follows by *i.h.*, and if it is internal to $L\langle f\rangle$, it is straightforward to close the diagram by resorting to the fact that $\equiv$ preserves the shape of $L\langle f\rangle$ (Lemma 23). The nontrivial case is when the $\Leftrightarrow$ step overlaps $F'\langle t'\rangle$ and $L\langle f\rangle$. There are two cases:

a) *The substitution comes from* t'. That is, $F' = \langle\cdot\rangle$ and t' has a substitution at its root. Then t' must be a $\mapsto_{\mathtt{e}}$-redex $t' = V''\langle x\rangle[x{\leftarrow}L\langle f\rangle]$. The diagram then is the same as in case 2b, reading it bottom-up.

b) *The substitution comes from* F'. That is: $F' = V''[x{\leftarrow}r]$ and the rewriting step is internal to $V''\langle t'\rangle$, reducing it to w', *i.e.* $w = w'[x{\leftarrow}r]$. The proof is then straightforward:

$$\begin{array}{ccc} V''\langle t'\rangle[x{\leftarrow}r]\, L\langle f\rangle & \xrightarrow{\qquad\qquad\circ} & w'[x{\leftarrow}r]\, L\langle f\rangle \\ \equiv_{@l} & & \equiv_{@l} \\ (V''\langle t'\rangle\, L\langle f\rangle)[x{\leftarrow}r] & \dashrightarrow & (w', L\langle f\rangle)[x{\leftarrow}r] \end{array}$$

c) *The substitution comes from* L. That is: $L = L'[x{\leftarrow}r]$. Then

$$\begin{array}{ccc} F'\langle t'\rangle\, L'\langle f\rangle[x{\leftarrow}r] & \xrightarrow{\qquad\qquad\circ} & w\, L'\langle f\rangle[x{\leftarrow}r] \\ \equiv_{@r} & & \equiv_{@r} \\ (F'\langle t'\rangle\, L'\langle f\rangle)[x{\leftarrow}r] & \dashrightarrow & (w, L'\langle f\rangle)[x{\leftarrow}r] \end{array}$$

4) **Inductive case 2: right of an application** $F = qF'$**.** The situation is:

$$t = q\, F'\langle t'\rangle \multimap q\, r = u$$

for some r. Reasoning as in the previous case (*left of an application*), if the $\Leftrightarrow$ step is internal to $F'\langle t'\rangle$, the result follows by *i.h.*, and it is immediate also if it is internal to q.
The remaining possibility is that the $\Leftrightarrow$ step overlaps with q or $F'\langle t'\rangle$. As in the previous case, this is only be possible because of a *commutation with application* rule. Cases:

a) *The substitution comes from* t'. That is, $F' = \langle\cdot\rangle$ and t' is a $\mapsto_{\mathtt{e}}$-redex $t' = V''\langle y\rangle[y{\leftarrow}L'\langle f\rangle]$. The

diagram then is the same as in case 2c, reading it bottom-up.

b) *The substitution comes from F'.* That is, $F' = V''[x{\leftarrow}w']$. This case is then straightforward:

$$\begin{array}{ccc} q\,V''\langle t'\rangle[x{\leftarrow}w'] & \longrightarrow\!\circ & q\,r[x{\leftarrow}w'] \\ \equiv_{@r} & & \equiv_{@r} \\ (q\,V''\langle t'\rangle)[x{\leftarrow}w'] & \dashrightarrow\!\circ & (q\,r)[x{\leftarrow}w'] \end{array}$$

c) *The substitution comes from q.* That is, $q = q'[x{\leftarrow}w']$. This case is straightforward:

$$\begin{array}{ccc} q'[x{\leftarrow}w']\,F'\langle t'\rangle & \longrightarrow\!\circ & q'[x{\leftarrow}w']\,r \\ \equiv_{@l} & & \equiv_{@l} \\ (q'\,F'\langle t'\rangle)[x{\leftarrow}w'] & \dashrightarrow\!\circ & (q'\,r)[x{\leftarrow}w'] \end{array}$$

5) **Inductive case 3: left of a substitution $F = F'[x{\leftarrow}q]$.** The situation is:

$$t = F'\langle t'\rangle[x{\leftarrow}q] \multimap r[x{\leftarrow}q] = u$$

If the $\Leftrightarrow$ step is internal to $F'\langle t'\rangle$, the result follows by *i.h.*. If it is internal to q, the steps are orthogonal, which makes the diagram trivial. The remaining possibility is that the substitution $[x{\leftarrow}q]$ is involved in the $\Leftrightarrow$ redex. By case analysis on the kind of the step $\equiv$:

a) *Commutation of independent substitutions $\equiv_{com}$.* Since $F'\langle t'\rangle$ must have a substitution at the root, there are two possibilities:

i) *The substitution comes from t'.* That is, $F' = \langle\cdot\rangle$ and t' is a $\mapsto_{\mathtt{e}}$-redex $t' = V''\langle y\rangle[y{\leftarrow}L\langle f\rangle]$, with $x \notin \mathtt{fv}(L\langle f\rangle)$. Then:

$$\begin{array}{ccc} V''\langle y\rangle[y{\leftarrow}L\langle f\rangle][x{\leftarrow}q] & \overset{\circ}{\longrightarrow}\!\circ & L\langle V''\langle f\rangle[y{\leftarrow}f]\rangle[x{\leftarrow}q] \\ \equiv_{com} & & \equiv^*_{com} \\ V''\langle y\rangle[x{\leftarrow}q][y{\leftarrow}L\langle f\rangle] & \overset{\circ}{\dashrightarrow}\!\circ & L\langle V''\langle f\rangle[x{\leftarrow}q][y{\leftarrow}f]\rangle \end{array}$$

ii) *The substitution comes from F'.* That is, $F' = V''[y{\leftarrow}w']$ with $x \notin \mathtt{fv}(w')$. This case is direct:

$$\begin{array}{ccc} V''\langle t'\rangle[y{\leftarrow}w'][x{\leftarrow}q] & \overset{\circ}{\longrightarrow}\!\circ & V''\langle u'\rangle[y{\leftarrow}w'][x{\leftarrow}q] \\ \equiv_{com} & & \equiv_{com} \\ V''\langle t'\rangle[x{\leftarrow}q][y{\leftarrow}w'] & \overset{\circ}{\dashrightarrow}\!\circ & V''\langle u'\rangle[x{\leftarrow}q][y{\leftarrow}w'] \end{array}$$

b) *Commutation with application $\equiv_{@}$.* $F'\langle t'\rangle$ must be an application. This allows for three possibilities:

i) *The application comes from t'.* That is, $F' = \langle\cdot\rangle$ and t' is a $\mapsto_{\mathtt{m}}$-redex $t' = L\langle ly.t''\rangle\,L'\langle f\rangle$. two sub-cases, whether $[x{\leftarrow}q]$ commutes on the left or on the right of the application. The left case is case 1b (read bottom-up), while the right case is case 1c (again bottom-up).

ii) *The application comes from $F' = V''\,L\langle w'\rangle$.* There are two sub-cases, whether $[x{\leftarrow}q]$ commutes on the left or on the right of the application. The left case is case 3b (read bottom-up), while the right case is case 3c (again bottom-up).

iii) *The application comes from $F' = q\,V''$.* Similarly to the previous case, it reduces to cases 4b and 4c.

c) *Composition of substitutions $\equiv_{[\cdot]}$.* Two sub-cases:

i) *The substitution comes from t'.* That is, $F' = \langle\cdot\rangle$ and t' is a $\mapsto_{\mathtt{e}}$-redex $t' = V''\langle y\rangle[y{\leftarrow}L\langle f\rangle]$, with $x \notin \mathtt{fv}(V''\langle y\rangle)$. Then:

$$\begin{array}{ccc} V''\langle y\rangle[y{\leftarrow}L\langle f\rangle][x{\leftarrow}q] & \overset{\circ}{\longrightarrow}\!\circ & L\langle V''\langle f\rangle[y{\leftarrow}f]\rangle[x{\leftarrow}q] \\ \equiv_{[\cdot]} & & = \\ V''\langle y\rangle[y{\leftarrow}L\langle f\rangle[x{\leftarrow}q]] & \overset{\circ}{\dashrightarrow}\!\circ & L\langle V''\langle f\rangle[y{\leftarrow}f]\rangle[x{\leftarrow}q] \end{array}$$

ii) *The substitution comes from F'.* That is, $F' = V''[y{\leftarrow}w']$ with $x \notin \mathtt{fv}(V''\langle t'\rangle)$. Then:

$$\begin{array}{ccc} V''\langle t'\rangle[y{\leftarrow}w'][x{\leftarrow}q] & \longrightarrow\!\circ & V''\langle u'\rangle[y{\leftarrow}w'][x{\leftarrow}q] \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ V''\langle t'\rangle[y{\leftarrow}w'[x{\leftarrow}q]] & \dashrightarrow\!\circ & V''\langle u'\rangle[y{\leftarrow}w'[x{\leftarrow}q]] \end{array}$$

■

A final lemma about the $\equiv$ relation will be useful later:

Lemma 24 (ES Commute with Evaluation Contexts via $\equiv$)**.** *Let S be a shallow context s.t. $x \notin \mathtt{fv}(S)$ and S doe not capture the variables in $\mathtt{fv}(u)$. Then $S\langle t[x{\leftarrow}u]\rangle \equiv S\langle t\rangle[x{\leftarrow}u]$.*

Proof: by induction on S.

1) *Empty Context $S = \langle\cdot\rangle$.* Then $S\langle t\rangle[x{\leftarrow}u] = t[x{\leftarrow}u] = S\langle t[x{\leftarrow}u]\rangle$.

2) *Application Left $S = S'w$.* Then

$$\begin{array}{lll} S\langle t[x{\leftarrow}u]\rangle & = & \\ S'\langle t[x{\leftarrow}u]\rangle w & = & \text{(by } \textit{i.h.}\text{)} \\ S'\langle t\rangle[x{\leftarrow}u]w & \equiv_{@l} & \\ (S'\langle t\rangle w)[x{\leftarrow}u] & = & \\ S\langle t\rangle[x{\leftarrow}u] & & \end{array}$$

3) *Application Right $F = wF'$.* Then

$$\begin{array}{lll} S\langle t[x{\leftarrow}u]\rangle & = & \\ wS'\langle t[x{\leftarrow}u]\rangle & = & \text{(by } \textit{i.h.}\text{)} \\ wS'\langle t\rangle[x{\leftarrow}u] & \equiv_{@r} & \\ (wS'\langle t\rangle)[x{\leftarrow}u] & = & \\ S\langle t\rangle[x{\leftarrow}u] & & \end{array}$$

4) *Substitution $F = F'[y{\leftarrow}w]$.* Then

$$\begin{array}{lll} S\langle t[x{\leftarrow}u]\rangle & = & \\ S'\langle t[x{\leftarrow}u]\rangle[y{\leftarrow}w] & = & \text{(by } \textit{i.h.}\text{)} \\ S'\langle t\rangle[x{\leftarrow}u][y{\leftarrow}w] & \equiv_{com} & \\ S'\langle t\rangle[y{\leftarrow}w][x{\leftarrow}u] & = & \\ S\langle t\rangle[x{\leftarrow}u] & & \end{array}$$

Note that $\equiv_{com}$ can be applied because of the hypotheses $x \notin \mathtt{fv}(S)$ and S doe not capture the variables in $\mathtt{fv}(u)$.

■

APPENDIX C
PROOFS OMITTED FROM SUBSECT. VII-A
(HIGH-LEVEL IMPLEMENTATION)

First, the High-Level Implementation Theorem.

Proof of Theorem 3 (page 6)

Proof: the proof is a minimal variation over the proof of Theorem 4.2, page 4, in [1]. Essentially we merged the trace and syntactic bound properties of that statement into our locally bound property. Note that for the global bound there is nothing to prove, it follows from the the hypothesis itself and projection. ■

Now, we prove that $(\to_{\mathtt{f}}, \multimap_{\mathtt{f}})$ is a high-level implementation system, *i.e.* Theorem 4.

The normal form property required for high-level implementation system has already been proved (Theorem 2). It only remains to prove the projection property.

Lemma 25 (Projection of a Rewriting Step)**.** *Let $t = F\langle u\rangle$ and F be an evaluation context.*

1) Multiplicative Projection*: if $t \multimap_{\mathtt{m}} w$ then $t{\downarrow} \to_{\mathtt{f}} w{\downarrow}$. More precisely, if $F\langle u\rangle \multimap_{\mathtt{m}} F\langle w\rangle$ with $u \mapsto_{\mathtt{m}} w$ then $F{\downarrow}\langle u{\downarrow_F}\rangle \to_{\mathtt{f}} F{\downarrow}\langle w{\downarrow_F}\rangle$ with $u{\downarrow_F} \mapsto_{\mathtt{f}} w{\downarrow_F}$;*
2) Exponential Projection*: if $t \multimap_{\mathtt{e}} w$ then $t{\downarrow} = w{\downarrow}$;*

Proof:

1) By induction on F. Cases:
   a) *Empty Context* $F = \langle\cdot\rangle$. Let $t = L\langle lx.r\rangle L'\langle f\rangle \mapsto_{\mathtt{m}} L\langle r[x\leftarrow L'\langle f\rangle]\rangle = w$. By induction on L. Two cases:
      i) *Empty context* $L = \langle\cdot\rangle$. Then $t = lx.rL'\langle f\rangle \mapsto_{\mathtt{m}} r[x\leftarrow L'\langle f\rangle] = w$

$$\begin{array}{lll} t{\downarrow} & = & \\ ((lx.r)L'\langle f\rangle){\downarrow} & = & \\ (lx.r{\downarrow})L'\langle f\rangle{\downarrow} & \to_{\mathtt{f}} & \\ r{\downarrow}\{x\leftarrow L'\langle f\rangle{\downarrow}\} & = & \\ r[x\leftarrow L'\langle f\rangle]{\downarrow} & = & \\ w{\downarrow} & & \end{array}$$

      ii) *Substitution* $L = L'[y\leftarrow q]$. Then $t = L\langle lx.r\rangle[y\leftarrow q]L'\langle f\rangle \mapsto_{\mathtt{m}} L\langle r[x\leftarrow L'\langle f\rangle]\rangle[y\leftarrow q] = w$. We have

$$\begin{array}{lll} t{\downarrow} & = & \\ (L\langle lx.r\rangle[y\leftarrow q]L'\langle f\rangle){\downarrow} & = & \\ L\langle lx.r\rangle[y\leftarrow q]{\downarrow}L'\langle f\rangle{\downarrow} & = & \\ L\langle lx.r\rangle{\downarrow}\{y\leftarrow q{\downarrow}\}L'\langle f\rangle{\downarrow} & = & \\ L\langle lx.r\rangle{\downarrow}L'\langle f\rangle{\downarrow}\{y\leftarrow q{\downarrow}\} & = & \\ (L\langle lx.r\rangle L'\langle f\rangle){\downarrow}\{y\leftarrow q{\downarrow}\} & \mapsto_{\mathtt{f}} & \textit{(i.h.} \text{ and Lemma 17)} \\ L\langle r[x\leftarrow L'\langle f\rangle]\rangle{\downarrow}\{y\leftarrow q{\downarrow}\} & = & \\ L\langle r[x\leftarrow L'\langle f\rangle]\rangle[y\leftarrow q]{\downarrow} & = & \\ w{\downarrow} & & \end{array}$$

   b) *Application Left* $F = F'L\langle f\rangle$. Then $F'\langle u\rangle L\langle f\rangle \multimap_{\mathtt{m}} F'\langle w\rangle L\langle f\rangle$ with $u \mapsto_{\mathtt{m}} w$. We have:

$$\begin{array}{lll} F\langle u\rangle{\downarrow} & = & \\ (F'\langle u\rangle L\langle f\rangle){\downarrow} & = & \\ F'\langle u\rangle{\downarrow}L\langle f\rangle{\downarrow} & \to_{\mathtt{f}} & \textit{(i.h.)} \\ F'\langle w\rangle{\downarrow}L\langle f\rangle{\downarrow} & = & \\ (F'\langle w\rangle L\langle f\rangle){\downarrow} & = & \\ F\langle w\rangle{\downarrow} & & \end{array}$$

      Actually, the $\to_{\mathtt{f}}$ step is justified by the *i.h.* and the fact that $\langle\cdot\rangle L\langle f\rangle{\downarrow}$ is an evaluation context because $L\langle f\rangle{\downarrow}$ is a fireball (by Lemma 19). The *i.h.* also gives $u{\downarrow_{F'}} \mapsto_{\mathtt{f}} w{\downarrow_{F'}}$. To conclude note that $u{\downarrow_F} = u{\downarrow_{F'f}} = u{\downarrow_{F'}} \mapsto_{\mathtt{f}} w{\downarrow_{F'}} = w{\downarrow_{F'f}} = w{\downarrow_F}$.
   c) *Application Right* $F = wF'$. Follows from the *i.h.*, along the lines of the previous case.
   d) *Substitution* $F = F'[x\leftarrow r]$. Then $F'\langle u\rangle[x\leftarrow r] \multimap_{\mathtt{m}} F'\langle w\rangle[x\leftarrow r]$ with $u \mapsto_{\mathtt{m}} w$. We have:

$$\begin{array}{lll} F\langle u\rangle{\downarrow} & = & \\ F'\langle u\rangle[x\leftarrow r]{\downarrow} & = & \\ F'\langle u\rangle{\downarrow}\{x\leftarrow r{\downarrow}\} & \to_{\mathtt{f}} & \textit{(i.h.} \text{ and Lemma 17)} \\ F'\langle w\rangle{\downarrow}\{x\leftarrow r{\downarrow}\} & = & \\ F'\langle w\rangle[x\leftarrow r]{\downarrow} & = & \\ F\langle w\rangle{\downarrow} & & \end{array}$$

      The *i.h.* also gives $u{\downarrow_{F'}} \mapsto_{\mathtt{f}} w{\downarrow_{F'}}$. To conclude note that

$$\begin{array}{lll} u{\downarrow_F} & = & \\ u{\downarrow_{F'[x\leftarrow r]}} & = & \\ u{\downarrow_{F'}}\{x\leftarrow r{\downarrow}\} & \mapsto_{\mathtt{f}} & \text{(Lemma 17)} \\ w{\downarrow_{F'}}\{x\leftarrow r{\downarrow}\} & = & \\ w{\downarrow_{F'[x\leftarrow r]}} & = & \\ w{\downarrow_F} & & \end{array}$$

2) We prove that if $t \mapsto_{\mathtt{e}} w$ then $t{\downarrow} = w{\downarrow}$ for any evaluation context F. From Lemma 30.3 the statement follows. By induction on F. We have $t = F'\langle x\rangle[x\leftarrow L\langle f\rangle] \mapsto_{\mathtt{e}} L\langle F'\langle f\rangle[x\leftarrow f]\rangle = w$. By induction on L. Two cases:
   a) *Empty context* $L = \langle\cdot\rangle$. Then $t = F'\langle x\rangle[x\leftarrow f] \mapsto_{\mathtt{e}} F'\langle f\rangle[x\leftarrow f] = w$

$$\begin{array}{lll} t{\downarrow} & = & \\ F'\langle x\rangle[x\leftarrow f]{\downarrow} & = & \\ F'\langle x\rangle{\downarrow}\{x\leftarrow f{\downarrow}\} & = & \\ F'{\downarrow}\langle x{\downarrow_{F'}}\rangle\{x\leftarrow f{\downarrow}\} & = & \text{(by Lemma 30.6)} \\ F'{\downarrow}\langle x\rangle\{x\leftarrow f{\downarrow}\} & = & \\ F'{\downarrow}\langle f{\downarrow}\rangle\{x\leftarrow f{\downarrow}\} & = & \\ F'{\downarrow}\langle f{\downarrow_{F'}}\rangle\{x\leftarrow f{\downarrow}\} & = & \\ F'\langle f\rangle{\downarrow}\{x\leftarrow f{\downarrow}\} & = & \\ F'\langle f\rangle[x\leftarrow f]{\downarrow} & & \\ w{\downarrow} & & \end{array}$$

   b) *Substitution* $L = L'[y\leftarrow q]$. Then $t = F'\langle x\rangle[x\leftarrow L\langle f\rangle[y\leftarrow q]] \mapsto_{\mathtt{e}} L\langle F'\langle f\rangle[x\leftarrow f]\rangle[y\leftarrow q] =$

w.

$$\begin{array}{lll} t\downarrow & = & \\ F'\langle x\rangle[x{\leftarrow}L\langle f\rangle[y{\leftarrow}q]]\downarrow & = & \\ F'\langle x\rangle\downarrow\{x{\leftarrow}L\langle f\rangle[y{\leftarrow}q]\downarrow\} & = & \\ F'\langle x\rangle\downarrow\{x{\leftarrow}L\langle f\rangle\downarrow\{y{\leftarrow}q\downarrow\}\} & = & \\ F'\langle x\rangle\downarrow\{x{\leftarrow}L\langle f\rangle\downarrow\}\{y{\leftarrow}q\downarrow\} & = & \\ F'\langle x\rangle[x{\leftarrow}L\langle f\rangle]\downarrow\{y{\leftarrow}q\downarrow\} & = & \text{(by } \textit{i.h.}\text{)} \\ L\langle F'\langle f\rangle[x{\leftarrow}f]\rangle\downarrow\{y{\leftarrow}q\downarrow\} & = & \\ L\langle F'\langle f\rangle[x{\leftarrow}f]\rangle[y{\leftarrow}q]\downarrow & = & \\ w\downarrow & & \end{array}$$

■

To prove that $(\to_{\mathtt{f}}, \multimap_{\mathtt{f}})$ is a high-level implementation system we only have to put together the various results.

Proof of Theorem 4 (page 6)

Proof: immediate from Corollary 5 and Lemma 25. ■

APPENDIX D
PROOFS OMITTED FROM SUBSECT. VII-B
(LOW-LEVEL IMPLEMENTATION: ABSTRACT MACHINES)

Proof of Lemma 1 (page 7)

Proof: straightforward induction on the length of $t\ (\to \cup \equiv)^*\ u$, using the strong bisimulation property. ■

Proof of Theorem 5 (page 7)

Proof: the proof can be found in [15] (Theorems 4.2 and 4.4) up to trivial modifications due to minor changes in the definition of distilleries and their properties. ■

APPENDIX E
PROOFS OMITTED FROM SECT. VIII
(AN INEFFICIENT DISTILLERY: THE GLAM MACHINE)

The aim of this section is to prove Theorem 7, *i.e.* that $(GLAM, \multimap_{\mathtt{f}}, \equiv, \underline{\cdot})$ is a reflective explicit distillery.

Proof of Lemma 2 (page 9)

Proof: by induction over the length of the execution. The base case holds because $\overline{t}$ is initial. The inductive step is by cases over the kind of transition. All the verifications are trivial inspections of the transition. ■

The first step to prove Theorem 7 is the distillation property. Note from the statement that the distillation is explicit (see Definition 3).

Lemma 26 (Explicit Distillation)**.** *Let s be a reachable state. Then:*

1) Commutative*: If $s \rightsquigarrow_{\mathtt{c}_{1,2,3}} s'$ then $\underline{s} = \underline{s'}$;*
2) Multiplicative*: If $s \rightsquigarrow_{\mathtt{m}} s'$ then $\underline{s} \multimap_{\mathtt{m}} \equiv \underline{s'}$;*
3) Exponential*: If $s \rightsquigarrow_{\mathtt{e}} s'$ then $\underline{s} \multimap_{\mathtt{e}} \underline{s'}$.*

Proof:

- Case $\rightsquigarrow_{\mathtt{c}_1}$:

$$\begin{array}{ll} \underline{(D, \overline{t}\overline{u}, \pi, E)} & = \\ \underline{\langle D\langle\langle\overline{t}\overline{u}\rangle\underline{\pi}\rangle\rangle E} & = \\ \underline{\langle D\langle\langle\overline{t}\langle\cdot\rangle\rangle\underline{\pi}\rangle\langle\overline{u}\rangle\rangle E} & = \\ \underline{\langle(\overline{t}, \pi) : D\langle\langle\overline{u}\rangle\rangle\rangle E} & = \\ \underline{\langle(\overline{t}, \pi) : D\langle\underline{\epsilon}\langle\overline{u}\rangle\rangle\rangle E} & = \\ \underline{((\overline{t}, \pi) : D, \overline{u}, \epsilon, E)} & \end{array}$$

- Case $\rightsquigarrow_{\mathtt{m}}$:

$$\begin{array}{lll} \underline{(D, lx.\overline{t}, \overline{u} : \pi, E)} & = & \\ \underline{\langle D\langle\overline{u} : \pi\langle lx.\overline{t}\rangle\rangle\rangle E} & = & \\ \underline{\langle D\langle\underline{\pi}\langle(lx.\overline{t})\overline{u}\rangle\rangle\rangle E} & \multimap_{\mathtt{m}} & \text{(by Lemma 2.3,4)} \\ \underline{\langle D\langle\underline{\pi}\langle\overline{t}[x{\leftarrow}\overline{u}]\rangle\rangle\rangle E} & \equiv & \text{(by Lemma 24)} \\ \underline{\langle D\langle\langle t\rangle\underline{\pi}\rangle[x{\leftarrow}\overline{u}]\rangle E} & = & \\ \underline{\langle D\langle\langle t\rangle\underline{\pi}\rangle\rangle[x{\leftarrow}\overline{u}] : E} & = & \\ \underline{(D, \overline{t}, \pi, [x{\leftarrow}\overline{u}] : E)} & & \end{array}$$

  Note that the multiplicative step is justified by points 3 and 4 of Lemma 2, for which $\overline{u}$ is a fireball and $\underline{\langle D\langle\underline{\pi}\rangle\rangle E}$ is an evaluation context. Moreover, the $\equiv$ step holds because by Lemma 2.1 (well-namedness) x occurs only in $\overline{t}$ and so by Lemma 24 the substitution $[x{\leftarrow}\overline{u}]$ commutes with the environment $\underline{D\langle\underline{\pi}\rangle}$.

- Case $\rightsquigarrow_{\mathtt{c}_2}$:

$$\begin{array}{ll} \underline{((\overline{t}, \pi) : D, lx.\overline{u}, \epsilon, E)} & = \\ \underline{\langle(\overline{t}, \pi) : D\langle\underline{\epsilon}\langle lx.\overline{u}\rangle\rangle\rangle E} & = \\ \underline{\langle D\langle\langle\overline{t}lx.\overline{u}\rangle\underline{\pi}\rangle\rangle E} & = \\ \underline{\langle D\langle lx.\overline{u} : \pi\langle\overline{t}\rangle\rangle\rangle E} & = \\ \underline{(D, \overline{t}, lx.\overline{u} : \pi, E)} & \end{array}$$

- Case $\rightsquigarrow_{\mathtt{c}_3}$:

$$\begin{array}{ll} \underline{((\overline{t}, \pi) : D, a, \pi', E)} & = \\ \underline{\langle(\overline{t}, \pi) : D\langle\underline{\pi'}\langle a\rangle\rangle\rangle E} & = \\ \underline{\langle D\langle\underline{\pi}\langle\overline{t}\langle a\rangle\underline{\pi'}\rangle\rangle\rangle E} & = \\ \underline{\langle D\langle\langle a\rangle\underline{\pi'} : \pi\langle\overline{t}\rangle\rangle\rangle E} & = \\ \underline{(D, \overline{t}, \langle a\rangle\underline{\pi'} : \pi, E)} & \end{array}$$

- Case $\rightsquigarrow_{\mathtt{e}}$:

$$\begin{array}{lll} \underline{(D, x, \pi, E_1[x{\leftarrow}\overline{u}]E_2)} & = & \\ \underline{\langle D\langle\langle x\rangle\underline{\pi}\rangle\rangle E_1[x{\leftarrow}\overline{u}]E_2} & = & \\ \underline{\langle\langle D\langle\langle x\rangle\underline{\pi}\rangle\rangle E_1[x{\leftarrow}\overline{u}]\rangle E_2} & \multimap_{\mathtt{e}} & \text{(by Lemma 2.3,4)} \\ \underline{\langle\langle D\langle\langle\overline{u}^{\alpha}\rangle\underline{\pi}\rangle\rangle E_1[x{\leftarrow}\overline{u}]\rangle E_2} & = & \\ \underline{\langle D\langle\langle\overline{u}^{\alpha}\rangle\underline{\pi}\rangle\rangle E_1[x{\leftarrow}\overline{u}]E_2} & = & \\ \underline{(D, \overline{u}^{\alpha}, \pi, E_1[x{\leftarrow}\overline{u}]E_2)} & & \end{array}$$

  Note that the exponential step is justified by points 3 and 4 of Lemma 2, for which $\overline{u}$ is a fireball and $\underline{E_2}$ and $\underline{E_1\langle D\langle\underline{\pi}\rangle\rangle}$ are evaluation contexts. ■

Next we prove progress. We first need to redefine the size of the machine state to ignore the new environment component:

Definition 9. $|(D, \overline{t}, \pi, E)| := |\overline{t}| + \Sigma_{(\overline{u},\pi)\in D}|\overline{u}|$

Lemma 27 (Termination)**.** *$\rightsquigarrow_{\mathtt{c}}$ is terminating*

Proof: just reuse the proof of Corollary **??**. ■

Lemma 28 (Determinism)**.** *The transition relation $\rightsquigarrow$ of the GLAM is deterministic.*

Proof: a simple inspection of the transitions show no critical pairs. ■

Lemma 29 (Progress)**.** *if s is reachable, $\mathtt{nf}_{\mathtt{c}}(s) = s$ and $\underline{s} \multimap_{\mathtt{x}} t$ with $\mathtt{x} \in \{\mathtt{sm}, \mathtt{se}\}$, then there exists s' such that $s \rightsquigarrow_{\mathtt{x}} s'$, i.e., s is not final.*

Proof: by Lemma 28 and Lemma 26 it is sufficient to show that every reachable stuck state decodes to a normal form. The only stuck forms are:

- (D, x, π, E) where x is not defined in E. The state is not reachable because it would violate the Closure invariant (Lemma 2.1).
- $(\epsilon, lx.\overline{t}, \epsilon, E)$ that decodes to $\langle lx.\overline{t}\rangle\underline{E}$, that by the contextual decoding invariant (Lemma 2.1) is a normal form.
- (ϵ, a, π, E) that decodes to $\langle\langle a\rangle\underline{\pi}\rangle\underline{E}$, that by the contextual decoding invariant (Lemma 2.1) is a normal form. ■

Proof of Theorem 7 (page 9)
Proof: it follows from Lemma 26 and Lemma 29. ■

APPENDIX F
PROOFS OMITTED FROM SECT. IX
(INTERLUDE II: RELATIVE UNFOLDINGS)

Proof of Lemma 3 (page 9)
Proof: by induction on W. ■

Lemma 30 (Properties of Relative Unfoldings)**.** *Let t and u be terms and S be a shallow context.*

1) Commutation*: $(\lambda x.t)\!\downarrow_S = \lambda x.t\!\downarrow_S$, $(tu)\!\downarrow_S = t\!\downarrow_S u\!\downarrow_S$, $t\{x\leftarrow u\}\!\downarrow_S = t\!\downarrow_S\{x\leftarrow u\!\downarrow_S\}$, $t\{x\leftarrow u\}\!\downarrow_S = t[x\leftarrow u]\!\downarrow_S$, $S\!\downarrow\{x\leftarrow t\!\downarrow\} = S\{x\leftarrow t\!\downarrow\}\!\downarrow$, and $t\!\downarrow_{S[x\leftarrow u]} = t\{x\leftarrow u\!\downarrow\}\!\downarrow_{S\{x\leftarrow u\downarrow\}}$.*
2) Freedom*: if S does not capture any free variable of t then $t\!\downarrow_S = t\!\downarrow$,*
3) Relativity*: if $t\!\downarrow = u\!\downarrow$ then $t\!\downarrow_S = u\!\downarrow_S$.*
4) Applicativity*: if S is applicative then $S\!\downarrow$ is applicative.*
5) Splitting*: $t\!\downarrow_{S\langle S'\rangle} = t\!\downarrow_{S'}\!\downarrow_S$.*
6) Factorisation*: $S'\langle t\rangle\!\downarrow_S = S'\!\downarrow_S\langle t\!\downarrow_{S\langle S'\rangle}\rangle$, in particular $S\langle t\rangle\!\downarrow = S\!\downarrow\langle t\!\downarrow_S\rangle$ and $L\langle t\rangle\!\downarrow_S = t\!\downarrow_{S\langle L\rangle}$.*

Proof: Routine inductions on S or S'. ■

APPENDIX G
PROOFS OMITTED FROM SECT. XI
(THE USEFUL FIREBALL CALCULUS)

Proof of Lemma 4 (page 11)
Proof:

1) By induction on the pair (number of ES in S,S). Cases of S:
   a) *Empty, i.e. $S = \langle\cdot\rangle$.* Then $S\!\downarrow = \langle\cdot\rangle\!\downarrow = \langle\cdot\rangle$ is an evaluation context.
   b) *Right Application, i.e. $S = tS'$.* Then $S\!\downarrow = t\!\downarrow S'\!\downarrow =_{i.h.} t\!\downarrow F$ is an evaluation context.
   c) *Left Application, i.e. $S = S't$ with $t\!\downarrow$ a fireball f.* Then $S\!\downarrow = S'\!\downarrow t\!\downarrow =_{i.h.} Ff$ is an evaluation context.
   d) *Substitution, i.e. $S = S'[x\leftarrow t]$ with $t\!\downarrow$ a fireball f and $S'\{x\leftarrow t\!\downarrow\}$ is evaluable*. Note that the number of ES in $S'\{x\leftarrow t\!\downarrow\}$ is strictly smaller than the number of ES in S, because $t\!\downarrow$ has no ES. Then by *i.h.* $S'\{x\leftarrow t\!\downarrow\}\!\downarrow$ is an evaluation context. Now, $S\!\downarrow = S'[x\leftarrow t]\!\downarrow = S'\!\downarrow\{x\leftarrow t\!\downarrow\} =_{L.30.1} S'\{x\leftarrow t\!\downarrow\}\!\downarrow$ which is an evaluation context.
2) By induction on the pair (number of ES in S,S). Cases of S:
   a) *Empty, i.e. $S = \langle\cdot\rangle$.* Directions
      i) $\Rightarrow$*, i.e. S is evaluable*. Nothing to prove.
      ii) $\Leftarrow$. Then S is evaluable.
   b) *Right Application, i.e. $S = wS'''$.* Note that S' cannot be empty, otherwise $S = wS''' = S''u$ and S would have two holes. Then $S' = wS_4$ for some S_4, and the statements follows from the *i.h.* applied to S''' and S_4.
   c) *Left Application, i.e. $S = S'''w$.* Directions:
      i) $\Rightarrow$*. Since S is evaluable, $w\!\downarrow$ is a fireball, and S''' is evaluable*. Note that either S' is empty, and then $u = w$ and the statement holds because $w\!\downarrow_{\langle\cdot\rangle} = w\!\downarrow$ is a fireball, or $S' = S_4w$ with S_4 s.t.—say—$S''' = S_4\langle S''[x\leftarrow u]\rangle$. Now, note that $u\!\downarrow_{S'} = u\!\downarrow_{S_4w} = u\!\downarrow_{S_4}$ and the statement follows by the *i.h.* applied to S'''.
      ii) $\Leftarrow$. By taking $S' := \langle\cdot\rangle$, the hypothesis becomes $w\!\downarrow_{\langle\cdot\rangle} = w\!\downarrow$ is a fireball. We are left to show that S''' is evaluable, that is given by the *i.h.*.
   d) *Substitution, i.e. $S' = S'''[y\leftarrow w]$.*
      i) $\Rightarrow$*. Since S is evaluable, $w\!\downarrow$ is a fireball, and $S'''\{y\leftarrow w\!\downarrow\}$ is evaluable*. Note that either S' is empty, and then $u = w$ and the statement holds because $w\!\downarrow_{\langle\cdot\rangle} = w\!\downarrow$ is a fireball, or $S' = S_4[y\leftarrow w]$ for some S_4 that is a prefix of S''', *i.e.* s.t. $S''' = S_4\langle S''u\rangle$ or $S''' = S_4\langle S''[x\leftarrow u]\rangle$. Let's say that $S''' = S_4\langle S''u\rangle$. Now, applying the *i.h.* to
      $$\begin{array}{ll} S'''\{y\leftarrow w\!\downarrow\} & = \\ S_4\langle S''u\rangle\{y\leftarrow w\!\downarrow\} & = \\ S_4\{y\leftarrow w\!\downarrow\}\langle S''\{y\leftarrow w\!\downarrow\}u\{y\leftarrow w\!\downarrow\}\rangle & \end{array}$$
      we obtain that $u\{y\leftarrow w\!\downarrow\}\!\downarrow_{S_4\{y\leftarrow w\downarrow\}}$ is a fireball. We conclude noting that $u\{y\leftarrow w\!\downarrow\}\!\downarrow_{S_4\{y\leftarrow w\downarrow\}} =_{Lemma\ 30.1} u\!\downarrow_{S_4[y\leftarrow w]} = u\!\downarrow_S$ (the other case, $S''' = S_4\langle S''[x\leftarrow u]\rangle$, uses the same reasoning).
      ii) $\Leftarrow$. By taking $S' := \langle\cdot\rangle$, the hypothesis becomes $w\!\downarrow_{\langle\cdot\rangle} = w\!\downarrow$ is a fireball. We are left

to show that $S'''\{y\leftarrow w\downarrow\}$ is evaluable, that is given by the *i.h.*.

■

The following technical lemma is very useful to decompose and construct evaluation contexts compositionally.

Lemma 31.

1) *if $S\langle S'\rangle$ is evaluable then S is evaluable and $S'\downarrow_S$ is an evaluation context.*
2) *if S is evaluable, $S'\downarrow_S$ is an evaluation context and S' is without ES then $S\langle S'\rangle$ is evaluable.*

Proof:

1) By induction on the pair (number of ES in S,S). Cases of S:
   a) *Empty, i.e.* $S = \langle\cdot\rangle$. The hypothesis becomes that S' is evaluable, and so $S'\downarrow_S = S'\downarrow_{\langle\cdot\rangle} = S'\downarrow$ is an evaluation context by Point 1. Clearly $\langle\cdot\rangle$ is evaluable.
   b) *Right Application, i.e.* $S = tS''$. By *i.h.*, S'' is evaluable, that implies S evaluable. Moreover, $S'\downarrow_S = S'\downarrow_{tS''} = S'\downarrow_{S''}$ which is an evaluation context by *i.h.*.
   c) *Left Application, i.e.* $S = S''u$. By *i.h.*, S'' is evaluable. From the hypothesis that $S\langle S'\rangle = S''\langle S'\rangle u$ is evaluable it follows that $u\downarrow$ is a fireball, and so S is evaluable. Moreover, $S'\downarrow_S = S'\downarrow_{S''u} = S'\downarrow_{S''}$ which is an evaluation context by *i.h.*.
   d) *Substitution, i.e.* $S = S''[x\leftarrow u]$. From the hypothesis that $S\langle S'\rangle = S''\langle S'\rangle[x\leftarrow u]$ is evaluable it follows that $u\downarrow$ is a fireball. Since $S''\{x\leftarrow u\downarrow\}$ has strictly less ES than S (because $u\downarrow$ has none), the *i.h.* gives that $S''\{x\leftarrow u\downarrow\}$ is evaluable, and so S is evaluable. Then $S'\downarrow_S = S'\downarrow_{S''[x\leftarrow u]} =_{L.30.1} S'\{x\leftarrow u\downarrow\}\downarrow_{S''\{x\leftarrow u\downarrow\}}$ that by *i.h.* is an evaluation context.
2) We prove that $u\downarrow_{S''}$ is a fireball whenever $S\langle S'\rangle = S''\langle S'''u\rangle$ or $S\langle S'\rangle = S''\langle S'''[x\leftarrow u]\rangle$, and conclude by applying Point 2. Now, since S' has no ES, if $S\langle S'\rangle = S''\langle S'''[x\leftarrow u]\rangle$ then $[x\leftarrow u]$ occurs in S, and S'' is a prefix of S. We obtain that $u\downarrow_{S''}$ is a fireball by applying Point 2 to S, that is evaluable by hypothesis. If $S\langle S'\rangle = S''\langle S'''u\rangle$ with S'' a prefix of S we reason similarly. Otherwise, the application $S'''u$ occurs in S', *i.e.* there is a context S_4 s.t. $S'' = S\langle S_4\rangle$ and $S' = S_4\langle S'''u\rangle$. Then we have $S'\downarrow_S = S_4\langle S'''u\rangle\downarrow_S =_{L.30.6} S_4\downarrow_S\langle(S'''u)\downarrow_{S\langle S_4\rangle}\rangle =_{L.30.1} S_4\downarrow_S\langle S'''\downarrow_{S\langle S_4\rangle}u\downarrow_{S\langle S_4\rangle}\rangle$, which by hypothesis is an evaluation context. Therefore, $u\downarrow_{S\langle S_4\rangle}$ is a fireball. We conclude with $u\downarrow_{S\langle S_4\rangle} =_{L.30.5} u\downarrow_{S_4}\downarrow_S =_{L.30.2} u\downarrow_S$, where the last equality follows because S_4, being a prefix of S', has no ES and so cannot capture the variables in u. ■

The next result to be proved is Theorem 8 ($(\rightarrow_{\mathtt{f}}, -\circ_{\mathtt{uf}})$ is a locally bounded high-level implementation system). We follow closely the same approach used for the Explicit FBC in Appendix B and Appendix C: first we define proper terms and the invariants of reduction; then we characterize normal forms; finally we prove projection and we obtain the theorem as a corollary.

Definition 10 (Proper Term)**.** *A term t is* proper *if*

1) Evaluability*: $t = S\langle u\rangle$ with S evaluable and u a l-term (without ES);*
2) Value*: no value in t contains ES.*

For instance, a proper term cannot have $-\circ_{\mathtt{um}}$ redexes inside ES.

Note that initial terms are proper and so the next lemma applies in particular when the starting term is initial.

Lemma 32 (Proper Invariant)**.** *Let t be a proper and closed term. If $t -\circ^*_{\mathtt{uf}} u$ then u is proper and closed.*

Proof: by induction on the length k of the derivation $t -\circ^*_{\mathtt{uf}} u$. The base case is trivial. For the step case, assume $t -\circ^{k-1}_{\mathtt{uf}} w -\circ_{\mathtt{uf}} u$. By *i.h.* w is proper and closed. We distinguish two cases:

1) *Case $w = S\langle L\langle lx.r\rangle q\rangle -\circ_{\mathtt{um}} S\langle L\langle r[x\leftarrow q]\rangle\rangle = u$ where $S\langle Lq\rangle$ is evaluable and applicative:*
   u is closed because w is. All values in u are values in the proper term w, and therefore they have no ES. Moreover r is a sub-term of a value of w, and therefore has no ES. Since $S\langle Lq\rangle$ is evaluable, $q\downarrow_S$ is a fireball by Lemma 4.2 and S and $S\langle L\rangle$ are evaluable by Lemma 31.1. Therefore $S\langle L\langle\langle\cdot\rangle[x\leftarrow q]\rangle\rangle$ is evaluable too by the other direction of Lemma 4.2 and the evaluability of $S\langle L\rangle$. Therefore u is proper.
2) *Case*
   $w = S\langle S'\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle -\circ_{\mathtt{ue}} S\langle L\langle S'\langle v\rangle[x\leftarrow v]\rangle\rangle = u$ where $S\langle S'[x\leftarrow L\langle v\rangle]\rangle$ is evaluable and applicative and $v\downarrow_{S\langle L\rangle} = \lambda y.p$:
   u is closed because w is. All values in u are values in the proper term w, and therefore they have no ES. In particular, v has no ES. Thus u is proper. ■

Lemma 33 (Normal Form Characterization)**.** *Let $t = S\langle u\rangle$ be a proper and closed term s.t. $u \not-\circ_{\mathtt{uf}}$ and S is evaluable. Then*

1) *either $u\downarrow_S$ is a fireball,*
2) *or $u\downarrow_S \rightarrow_{\mathtt{f}}$, more precisely exists S' s.t.*
   a) *$u = S'\langle x\rangle$ with*
   b) *$x \in \mathtt{fv}(u)$,*
   c) *$S\langle S'\rangle$ evaluable,*
   d) *$x\downarrow_S = \lambda y.w$, and*
   e) *S' is applicative.*

Proof: first of all, let us show that the conditions on S' imply $u\downarrow_S \rightarrow_{\mathtt{f}}$. We have:

$$\begin{array}{ll} u\downarrow_S & =_a \\ S'\langle x\rangle\downarrow_S & =_{L.30.6} \\ S'\downarrow_S\langle x\downarrow_{S\langle S'\rangle}\rangle & =_{c\&L.31.1} \\ F\langle x\downarrow_{S\langle S'\rangle}\rangle & =_{b\&L.30.2} \\ F\langle x\downarrow_S\rangle & =_d \\ F\langle\lambda y.w\rangle & \end{array}$$

and $F := S'\downarrow_S$ is applicative, by c and L.30.4. Then note that Point 1 and Point 2 are mutually exclusive. Indeed, by Lemma 15, an unfolded term which is a fireball is $\to_{\mathtt{f}}$-normal. So, if Point 1 holds then Point 2 does not, and vice-versa. Therefore, in the following proof for we only have to prove that Point 1 or Point 2 holds.

By induction on u. Cases:

1) *Variable* x. Since t is proper and closed, $u\downarrow_S$ is a fireball.
2) *Symbol* and *Abstraction*. Note that by properness, the abstraction is an ordinary λ-term, *i.e.* it does not contain ES. Then in both cases we can apply Lemma 19, obtaining that $u\downarrow_S$ is a fireball.
3) *Application* $u = wr$. Since $S\langle w\langle\cdot\rangle\rangle$ is an evaluable context, we can apply the *i.h.* to r, ending in one of the following two cases:
   a) $r\downarrow_S$ *is a fireball*. Then $S\langle\langle\cdot\rangle r\rangle$ is an evaluable context and we can apply the *i.h.* to w obtaining two cases:
      i) $w\downarrow_S$ *is a fireball*. Two kinds of fireball:
         - $w\downarrow_{S\langle\langle\cdot\rangle r\rangle}$ *is a inert* A. Then $u\downarrow_S = Ar\downarrow_S$ is a inert, *i.e.* a fireball.
         - $w\downarrow_{S\langle\langle\cdot\rangle r\rangle}$ *is an abstraction* $\lambda y.q$. Then $u\downarrow_S$ reduces, indeed $u\downarrow_S = w\downarrow_{S\langle\langle\cdot\rangle r\rangle} r\downarrow_S = (\lambda y.q)r\downarrow_S \to_{\mathtt{f}}$. In terms of contexts, note that w is not itself an abstraction, otherwise u would be a $\multimap_{\mathtt{um}}$-redex, *i.e.* w has the form $L\langle x\rangle$. Moreover, L does not capture x, otherwise $u = wr = L\langle y\rangle r$ would have a $\multimap_{\mathtt{ue}}$-redex (because t is proper and so the substitution on x in L can fire). Then $x \in \mathtt{fv}(w)$ (and so $x \in \mathtt{fv}(u)$) and $S' := Lr$ satisfies points a, b, d, e of the statement. For c, we only have to show that the content of every substitution in L unfolds to a fireball in its context (by Lemma 4.2). Note that, since t is proper, there is an evaluable context containing all the ES in t, *i.e.* the content of *every* substitution in t unfolds in its context to a fireball.
      ii) $w\downarrow_S$ *reduces, i.e.* $w\downarrow_S \to_{\mathtt{f}}$. We have $u\downarrow_S = w\downarrow_{S\langle\langle\cdot\rangle r\rangle} r\downarrow_S = w\downarrow_S r\downarrow_S \to_{\mathtt{f}}$ because $r\downarrow_S$ is a fireball and so $\langle\cdot\rangle r\downarrow_S$ is an evaluation context. In terms of contexts, set $S' := S''r$, where S'' is he context given by the *i.h.*. It is easily seen that S' satisfies the statement.
   b) $r\downarrow_S$ *reduces, i.e.* $r\downarrow_S \to_{\mathtt{f}}$. We have $u\downarrow_S = w\downarrow_{S\langle\langle\cdot\rangle r\rangle} r\downarrow_S$ because $w\downarrow_{S\langle\langle\cdot\rangle r\rangle}\langle\cdot\rangle$ is an evaluation context. In terms of contexts, set $S' := wS''$, where S'' is he context given by the *i.h.*. It is easily seen that S' satisfies the statement.
4) *Substitution* $u = w[x\leftarrow r]$. $S\langle\langle\cdot\rangle[x\leftarrow r]\rangle\downarrow$ is an evaluation context and we can apply the *i.h.* to w. Note that since $w\downarrow_{S\langle\langle\cdot\rangle[x\leftarrow r]\rangle} =_{L.30.6} w[x\leftarrow r]\downarrow_S = u\downarrow_S$, this case reduces to the *i.h.*. In terms of contexts (for Point 2), note that the context S'' given by the *i.h.* cannot expose an occurrence of x, otherwise there would be a $\multimap_{\mathtt{ue}}$-redex in u (because t is proper and so r has the form $L\langle v\rangle$). Thus, the context $S' := S''[x\leftarrow r]$ is easily seen to satisfy the statement (inheriting the properties of S'').

■

Corollary 6 (Normal Forms Unfold to Normal Forms)**.** *Let t be a closed proper term. If t is $\multimap_{\mathtt{uf}}$-normal then $t\downarrow$ is $\to_{\mathtt{f}}$-normal.*

Proof: note that applying Lemma 33 with $S := \langle\cdot\rangle$ and $u := t$ one obtains that $t\downarrow$ is a fireball, because the second case cannot happen, given that u now is closed and so it cannot be written as $u = S'\langle x\rangle$ with $x \in \mathtt{fv}(u)$. By Lemma 15, $t\downarrow$ is $\to_{\mathtt{f}}$-normal. ■

To prove the projection lemma we need to prove first as a technical lemma another sufficient condition for a context to be evaluable. The condition is based on the definition of position of a redex.

The *position* of a redex is (the context S exposing) the application that makes applicative the evaluable context in the side condition. For a $\multimap_{\mathtt{um}}$-redex, it is given by S, while for a $\multimap_{\mathtt{ue}}$-redex one needs to do a case analysis, because the application may lie in S or in S'. Note that such a notion of position for $\multimap_{\mathtt{ue}}$-redexes is different with respect to the one used in Subsect. B-A.

Lemma 34. *If $S\langle t\rangle$ has a redex having its position in t then S is evaluable.*

Proof: then the position of the redex has the form $S\langle S'\rangle$ for some context S'. By the hypothesis on redexes and Lemma 31.1, $S\langle S'\rangle$ is evaluable. By Lemma 31.1, S is evaluable. ■

Lemma 35 (Projection)**.** *Let $t = S\langle u\rangle \multimap_{\mathtt{uf}} S'\langle w\rangle = r$ by reducing a redex whose position lies in u. If the redex is*

1) Multiplicative*: then $u\downarrow_S \to_{\mathtt{f}} w\downarrow_{S'}$ and $t\downarrow \to_{\mathtt{f}} r\downarrow$;*
2) Exponential*: then $u\downarrow_S \to_{\mathtt{f}}$ and $t\downarrow = r\downarrow \to_{\mathtt{f}}$.*

In both cases $u\downarrow_S$ is not a fireball.

Proof: the fact that in both cases $u\downarrow_S$ is not a fireball, follows from Lemma 15 and the fact that $u\downarrow_S$ reduces. Cases:

1) *Multiplicative*. Note that in this case $S' = S$. Then $t\downarrow \to_{\mathtt{f}} r\downarrow$ follows from $u\downarrow_S \to_{\mathtt{f}} w\downarrow_S$. By Lemma 34, S is evaluable, and by Lemma 4.1 $S\downarrow$ is an evaluation context, so:

$$\begin{array}{ll} t\downarrow & = \\ S\langle u\rangle\downarrow & =_{L.30.6} \\ S\downarrow\langle u\downarrow_S\rangle & \to_{\mathtt{f}} \\ S\downarrow\langle w\downarrow_{S'}\rangle & =_{L.30.6} \\ S\langle w\rangle\downarrow & = \\ r\downarrow & \end{array}$$

   We now show $u\downarrow_S \to_{\mathtt{f}} w\downarrow_S$. Since the redex lies in u, we have $u = S'\langle L\langle\lambda x.w\rangle r\rangle$ and $t = S\langle S'\langle L\langle\lambda x.w\rangle r\rangle\rangle$ with $S\langle S'\langle\langle\cdot\rangle r\rangle\rangle$, and thus $S\langle S'\rangle$, evaluable. Moreover,

by Lemma 31.1 $r\downarrow_{S\langle S'\langle L\langle\lambda x.w\rangle\langle\cdot\rangle\rangle\rangle} = r\downarrow_{S\langle S'\rangle}$ is a fireball and $S'\downarrow_S$ is an evaluation context. Then

$$\begin{array}{lll} u\downarrow_S & = & \\ S'\langle L\langle\lambda x.w\rangle r\rangle\downarrow_S & = & \text{(Lemma 30.6)} \\ S'\downarrow_S\langle(L\langle\lambda x.w\rangle r)\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.1)} \\ S'\downarrow_S\langle L\langle\lambda x.w\rangle\downarrow_{S\langle S'\rangle} r\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.6)} \\ S'\downarrow_S\langle(\lambda x.w)\downarrow_{S\langle S'\langle L\rangle\rangle} r\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.1)} \\ S'\downarrow_S\langle\lambda x.w\downarrow_{S\langle S'\langle L\rangle\rangle} r\downarrow_{S\langle S'\rangle}\rangle & \to_{\mathtt{f}} & (S'\downarrow_S \text{ is an ev. cont. \& } r\downarrow_{S\langle S'\rangle} \text{ is a fireball}) \\ S'\downarrow_S\langle w\downarrow_{S\langle S'\langle L\rangle\rangle}\{x\leftarrow r\downarrow_{S\langle S'\rangle}\}\rangle & = & \text{(Lemma 30.1)} \\ S'\downarrow_S\langle w\{x\leftarrow r\}\downarrow_{S\langle S'\langle L\rangle\rangle}\rangle & = & \text{(Lemma 30.1)} \\ S'\downarrow_S\langle w[x\leftarrow r]\downarrow_{S\langle S'\langle L\rangle\rangle}\rangle & = & \text{(Lemma 30.6)} \\ S'\downarrow_S\langle L\langle w[x\leftarrow r]\rangle\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.6)} \\ S'\langle L\langle w[x\leftarrow r]\rangle\rangle\downarrow_S & = & \\ w\downarrow_S & & \end{array}$$

2) *Exponential*. We take $t\downarrow = r\downarrow$ for granted, because a substitution step by definition does not change the unfolding. Similarly to the previous case, $t\downarrow\to_{\mathtt{f}}$ follows from $u\downarrow_S\to_{\mathtt{f}}$. Indeed, by Lemma 31.1, S is evaluable, and by Lemma 4.1 $S\downarrow$ is an evaluation context, so:

$$t\downarrow = S\langle u\rangle\downarrow =_{L.30.6} S\downarrow\langle u\downarrow_S\rangle \to_{\mathtt{f}}$$

Now we prove $u\downarrow\to_{\mathtt{f}}$. We have $u = S'\langle L\langle x\rangle r\rangle$ and $t = S\langle S'\langle L\langle x\rangle r\rangle\rangle$. In t there is somewhere (in L, S', or S) a substitution $[x\leftarrow L'\langle q\rangle]$ with the hypothesis that q relatively unfolds to some value $\lambda y.w$ in its context. So, $x\downarrow_{S\langle S'\langle L\rangle\rangle} = \lambda y.w$. Moreover, by hypothesis $S\langle S'\rangle$ is evaluable, and so by Lemma 31.1 $S'\downarrow_S$ is an evaluation context. Finally, $r\downarrow_{S\langle S'\rangle}$ is a fireball, because $S\langle S'\langle\langle\cdot\rangle r\rangle\rangle$ is evaluable. Then

$$\begin{array}{lll} u\downarrow_S & = & \\ S'\langle L\langle x\rangle r\rangle\downarrow_S & = & \text{(Lemma 30.6)} \\ S'\downarrow_S\langle(L\langle x\rangle r)\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.1)} \\ S'\downarrow_S\langle L\langle x\rangle\downarrow_{S\langle S'\rangle} r\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.6)} \\ S'\downarrow_S\langle x\downarrow_{S\langle S'\langle L\rangle\rangle} r\downarrow_{S\langle S'\rangle}\rangle & = & (x\downarrow_{S\langle S'\langle L\rangle\rangle} = \lambda y.w) \\ S'\downarrow_S\langle(\lambda y.w) r\downarrow_{S\langle S'\rangle}\rangle & \to_{\mathtt{f}} & (S'\downarrow_S \text{ is an ev. cont. \& } r\downarrow_{S\langle S'\rangle} \text{ a fireball}) \\ S'\downarrow_S\langle w[y\leftarrow r\downarrow_{S\langle S'\rangle}]\rangle & & \end{array}$$

■

Determinism of $-\circ_{\mathtt{uf}}$ is the last ingredient to prove that $(\to_{\mathtt{f}}, -\circ_{\mathtt{uf}})$ is a locally bounded high-level implementation system.

Lemma 36 (Determinism)**.** *Let t be a term and $S\langle S_1\rangle$ and $S\langle S_2\rangle$ positions of $-\circ_{\mathtt{uf}}$-redexes. Then $S_1 = S_2$.*

Proof: by induction on S_1. Cases:

1) *Empty* $S_1 = \langle\cdot\rangle$. Cases:
   a) *Multiplicative Redex, i.e.* $u = L\langle\lambda x.r\rangle q$ with $q\downarrow_S$ a fireball. Now, S_2 cannot lie in $L\langle\lambda x.r\rangle$, otherwise by Lemma 35 $L\langle\lambda x.r\rangle\downarrow_{S\langle S_2\rangle}$ would not be a fireball, while by (properness and) Lemma 19 it does. Nor S_2 can lie in q, otherwise again by Lemma 35 $q\downarrow_S$ would not be a fireball. Then necessarily $S_2 = S_1 = \langle\cdot\rangle$.
   b) *Exponential Redex, i.e.* $u = S'\langle L\langle x\rangle r\rangle$. Now, S_2 cannot lie in $L\langle x\rangle$, otherwise by Lemma 35 $L\langle x\rangle\downarrow_{S\langle S_2\rangle}$ would not be a fireball, while by the hypothesis on the $-\circ_{\mathtt{ue}}$-redex it does (it is an abstraction). Nor S_2 can lie in r, otherwise again by Lemma 35 $r\downarrow_S$ would not be a fireball, while by the hypothesis on the $-\circ_{\mathtt{ue}}$-redex it does. Then necessarily $S_2 = S_1 = \langle\cdot\rangle$.
2) *Right Application* $S_1 = rS_1'$ and $t = rS_1'\langle q\rangle$. By Lemma 35, $S_1'\langle q\rangle\downarrow_{S\langle r\langle\cdot\rangle\rangle}$ has a $\to_{\mathtt{f}}$-redex and it is not a fireball, so no redexes can lie to its left, in particular S_2 does not lie in r. By Lemma 35, $S_1'\langle q\rangle\downarrow_{S\langle r\langle\cdot\rangle\rangle}$ is not a fireball, and so S_2 cannot be empty (*i.e.* $rS_1'\langle q\rangle$ cannot be the position of a $-\circ_{\mathtt{um}}$-redex). Then, $S_2 = uS_2'$, and the statement follows from the *i.h.* applied to S_1' and S_2'.
3) *Left Application* $S_1 = S_1'q$ and $t = S_1'\langle r\rangle q$. Note that S_2 cannot lie in q, otherwise by Lemma 35 $q\downarrow_{S\langle S_1'\langle r\rangle\langle\cdot\rangle\rangle}$ has a $\to_{\mathtt{f}}$-redex and it is not a fireball, and so no redexes—in particular the one of position $S\langle S_1\rangle$—can lie to its left, absurd. And S_2 cannot be empty (*i.e.* the position of a $-\circ_{\mathtt{um}}$-redex), because then $S_1'\langle r\rangle$ would have the form $L\langle\lambda x.p\rangle$, which by Lemma 35 cannot contain the position of a redex, because by Lemma 19 $L\langle\lambda x.p\rangle\downarrow_{S\langle\langle\cdot\rangle q\rangle}$ is a fireball. Then, $S_2 = S_2'w$, and the statement follows from the *i.h.* applied to S_1' and S_2'.
4) *Substitution* $S_1 = S_1'[x\leftarrow w]$. Then necessarily $S_2 = S_2'[x\leftarrow w]$ (remember the position of a $-\circ_{\mathtt{ue}}$-redex is an application) and the statement follows from the *i.h.*. ■

Proof of Theorem 8 (page 11)

Proof: the pair $(\to_{\mathtt{f}}, -\circ_{\mathtt{uf}})$ is an high-level implementation system because of Lemma 36, Lemma 35 and Corollary 6.

We deduce that the implementation system is locally bounded from the corresponding bound (Lemma 5.5) on the abstract machine that implements the calculus. An alternative, direct proof without any reference to abstract machines is surely possible, but we would need to establish first additional invariants on the ES that occur in the term. Intuitively, anyway, the local bound follows mainly from acyclicity of the explicit substitutions and the fact that only multiplicative steps can create a new ES, while exponential steps never duplicate terms containing ES. ■

Proof of Proposition 4 (page 11)

Proof: omitted. All postponement proofs are similar and lengthy. In Subsect. B-B of the Appendix we proved the lemma for the Explicit FBC. Other examples can be found in the long version of [15]. ■

A. Proofs Omitted From Sect. XII (The GLAMoUr Machine)

The aim of this section is to prove Theorem 9 ($(GLAMoUr, \multimap_{\mathtt{uf}}, \equiv, \underline{\cdot})$ is a reflective explicit distillery) and Theorem 10 (the useful implementation has bilinear low level and quadratic high level complexity). We start by proving that the invariants of the machine holds.

Lemma 37 (Contextual Decoding)**.** *$\underline{E}$ is a substitution context; $\underline{D}$ and $\underline{\pi}$ are shallow contexts without ES.*

Proof: by induction on E, D and π. ■

Proof of Lemma 5 (page 12)

Proof: by induction over the length of the execution. The base case holds because $\overline{t}$ is initial. The inductive step is by cases over the kind of transition. All the verifications are trivial apart for Point 4. For Point 4, evaluability for $\underline{E}$, $\underline{D}{\downarrow_E}$, $\underline{\pi}{\downarrow_E}$ follows from Point 3 and Lemma 4.2, while evaluability for $\langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E}$ follows from them and Lemma 4.2. ■

Lemma 38 (Explicit Distillation)**.** *Let s be a reachable state. Then:*

1) Commutative*: if $s \rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}} s'$ then $\underline{s} = \underline{s'}$;*
2) Multiplicative*: if $s \rightsquigarrow_{\mathtt{um}} s'$ then $\underline{s} \multimap_{\mathtt{um}} \equiv \underline{s'}$;*
3) Exponential*: if $s \rightsquigarrow_{\mathtt{ue}} s'$ then $\underline{s} \multimap_{\mathtt{ue}} \underline{s'}$.*

Proof: we list the transition in the order they appear in the definition of the machine.

- Case $(D, \overline{t}\overline{u}, \pi, E) \rightsquigarrow_{\mathtt{c}_1} (D : (\overline{t}, \pi), \overline{u}, \epsilon, E)$:

$$\begin{array}{ll} \underline{(D, \overline{t}\overline{u}, \pi, E)} & = \\ \underline{\langle \underline{D}\langle\langle \overline{t}\overline{u}\rangle\underline{\pi}\rangle\rangle\underline{E}} & = \\ \underline{\langle \underline{D}\langle\langle \overline{t}\langle\overline{u}\rangle\rangle\underline{\pi}\rangle\rangle\underline{E}} & = \\ \underline{\langle D : (\overline{t}, \pi)\langle\overline{u}\rangle\rangle\underline{E}} & = \\ \underline{\langle D : (\overline{t}, \pi)\langle\langle\overline{u}\rangle\underline{\epsilon}\rangle\rangle\underline{E}} & = \\ \underline{(D : (\overline{t}, \pi), \overline{u}, \epsilon, E)} & \end{array}$$

- Case $(D, lx.\overline{t}, \phi^l : \pi, E) \rightsquigarrow_{\mathtt{um}} (D, \overline{t}, \pi, [x{\leftarrow}\phi^l]E)$:

$$\begin{array}{lll} \underline{(D, lx.\overline{t}, \phi^l : \pi, E)} & = & \\ \underline{\langle \underline{D}\langle\langle lx.\overline{t}\rangle\underline{\phi^l : \pi}\rangle\rangle\underline{E}} & = & \\ \langle \underline{D}\langle\langle(lx.\overline{t})\underline{\phi}\rangle\underline{\pi}\rangle\rangle\underline{E} & \multimap_{\mathtt{um}} & \text{(by Lemma 5.4 and Lemma 5.3)} \\ \langle \underline{D}\langle\langle\overline{t}[x{\leftarrow}\phi]\rangle\underline{\pi}\rangle\rangle\underline{E} & \equiv & \text{(by Lemma 24)} \\ \langle \underline{D}\langle\langle\overline{t}\rangle\underline{\pi}\rangle[x{\leftarrow}\phi]\rangle\underline{E} & = & \\ \langle \underline{D}\langle\langle\overline{t}\rangle\underline{\pi}\rangle\rangle\underline{[x{\leftarrow}\phi^l]E} & = & \\ \underline{(D, \overline{t}, \pi, [x{\leftarrow}\phi^l]E)} & & \end{array}$$

- Case $(D : (\overline{t}, \pi), lx.\overline{u}, \epsilon, E) \rightsquigarrow_{\mathtt{c}_2} (D, \overline{t}, (lx.\overline{u})^v : \pi, E)$:

$$\begin{array}{ll} \underline{(D : (\overline{t}, \pi), lx.\overline{u}, \epsilon, E)} & = \\ \underline{\langle D : (\overline{t}, \pi)\langle\langle lx.\overline{u}\rangle\underline{\epsilon}\rangle\rangle\underline{E}} & = \\ \underline{\langle D : (\overline{t}, \pi)\langle lx.\overline{u}\rangle\rangle\underline{E}} & = \\ \underline{\langle \underline{D}\langle\langle\overline{t}(lx.\overline{u})\rangle\underline{\pi}\rangle\rangle\underline{E}} & = \\ \underline{\langle \underline{D}\langle\langle\overline{t}\rangle(lx.\overline{u})^v : \pi\rangle\rangle\underline{E}} & = \\ \underline{(D, \overline{t}, (lx.\overline{u})^v : \pi, E)} & \end{array}$$

- Case $(D : (\overline{t}, \pi), a, \pi', E) \rightsquigarrow_{\mathtt{c}_3} (D, \overline{t}, (a, \pi')^A : \pi, E)$:

$$\begin{array}{ll} \underline{(D : (\overline{t}, \pi), a, \pi', E)} & = \\ \underline{\langle D : (\overline{t}, \pi)\langle\langle a\rangle\underline{\pi'}\rangle\rangle\underline{E}} & = \\ \underline{\langle \underline{D}\langle\langle\overline{t}\langle a\rangle\underline{\pi'}\rangle\underline{\pi}\rangle\rangle\underline{E}} & = \\ \underline{\langle \underline{D}\langle\langle\overline{t}\rangle(a, \pi')^A : \pi\rangle\rangle\underline{E}} & = \\ \underline{(D, \overline{t}, (a, \pi')^A : \pi, E)} & \end{array}$$

- Case $(D : (\overline{t}, \pi), x, \pi', E_1[x{\leftarrow}\phi^A]E_2) \rightsquigarrow_{\mathtt{c}_4} (D, \overline{t}, (x, \pi')^A : \pi, E_1[x{\leftarrow}\phi^A]E_2)$:

$$\begin{array}{ll} \underline{(D : (\overline{t}, \pi), x, \pi', E_1[x{\leftarrow}\phi^A]E_2)} & = \\ \cdots & = \\ \underline{(D, \overline{t}, (x, \pi')^A : \pi, E_1[x{\leftarrow}\phi^A]E_2)} & \end{array}$$

  The proof is the one for the previous case $\rightsquigarrow_{\mathtt{c}_3}$, by replacing a with x and instantiating E with $E_1[x{\leftarrow}\phi^A]E_2$.
- Case $(D : (\overline{t}, \pi), x, \epsilon, E_1[x{\leftarrow}\overline{u}^v]E_2) \rightsquigarrow_{\mathtt{c}_5} (D, \overline{t}, x^v : \pi, E_1[x{\leftarrow}\overline{u}^v]E_2)$:

$$\begin{array}{ll} \underline{(D : (\overline{t}, \pi), x, \epsilon, E_1[x{\leftarrow}\overline{u}^v]E_2)} & = \\ \cdots & = \\ \underline{(D, \overline{t}, x^v : \pi, E_1[x{\leftarrow}\overline{u}^v]E_2)} & \end{array}$$

  The proof is the one for the previous case $\rightsquigarrow_{\mathtt{c}_4}$, by replacing $(lx.\overline{u})$ with x and instantiating E with $E_1[x{\leftarrow}\overline{u}^v]E_2$.
- Case $(D, x, \phi^l : \pi, E_1[x{\leftarrow}\overline{v}^v]E_2) \rightsquigarrow_{\mathtt{oes}} (D, \overline{v}^\alpha, \phi^l : \pi, E_1[x{\leftarrow}\overline{v}^v]E_2)$:

$$\begin{array}{lll} \underline{(D, x, \phi^l : \pi, E_1[x{\leftarrow}\overline{v}^v]E_2)} & = & \\ \underline{\langle \underline{D}\langle\langle x\rangle\underline{\phi^v : \pi}\rangle\rangle\underline{E_1[x{\leftarrow}\overline{v}^v]E_2}} & \multimap_{\mathtt{ue}} & \text{(by Lemma 5.4 and Lemma 5.3)} \\ \underline{\langle \underline{D}\langle\langle\overline{v}^\alpha\rangle\underline{\phi^l : \pi}\rangle\rangle E_1[x{\leftarrow}\overline{v}^v]E_2} & = & \\ \underline{(D, \overline{v}^\alpha, \phi^l : \pi, E_1[x{\leftarrow}\overline{v}^v]E_2)} & & \end{array}$$

■

The next lemma extends the notion of state size $|s|$ given in Definition 9 by ignoring the new machine component E. The precise definition is Definition 6.

Lemma 39 (Determinism)**.** *The transition relation $\rightsquigarrow$ of the GLAMoUr is deterministic.*

Proof: a simple inspection of the transitions shows no critical pairs. ■

Lemma 40 (Progress)**.** *if s is reachable, $\mathtt{nf}_{\mathtt{c}}(s) = s$ and $\underline{s} \multimap_{\mathtt{x}} t$ with $\mathtt{x} \in \{\mathtt{um}, \mathtt{ue}\}$, then there exists s' such that $s \rightsquigarrow_{\mathtt{x}} s'$, i.e., s is not final.*

Proof: by Lemma 39 and Lemma 38 it is sufficient to show that every reachable stuck state decodes to a normal form. The only stuck forms are:

- *Error states*. The state con only be (D, x, π, E) where x is not defined in E or it is defined to be a $\overline{t}^v$ where $\overline{t}$ is not a variable or a value.
  The state is not reachable because it would violate either the invariant in Lemma 5.1 or the invariant in Lemma 5.3.
- *Final states*. Cases:
  1) *The result is/unfolds to a value*. The state is $(\epsilon, \overline{t}, \epsilon, E)$ with $\overline{t}$ an abstraction or a variable bound

in E to a ϕ^v. By Lemma 37, $\underline{(\epsilon, \overline{t}, \epsilon, E)} = \langle \overline{t} \rangle \underline{E} = L\langle \overline{t} \rangle$ for some L. Note that $L\langle \overline{t} \rangle \downarrow = \overline{t}\downarrow_L$ is a fireball, indeed if $\overline{t}$ is an abstraction it is given by Lemma 19 and if it as a variable it is given by Lemma 5.3. Thus by Lemma 35, $L\langle \overline{t} \rangle$ is normal.

2) *The result is/unfolds to a inert.* The state is $(\epsilon, \overline{t}, \pi, E)$ with $\overline{t}$ a symbol a or a variable bound in E to a ϕ^A.
By Lemma 37, $\underline{(\epsilon, \overline{t}, \pi, E)} = \langle \langle \overline{t} \rangle \underline{\pi} \rangle \underline{E} = L\langle \langle \overline{t} \rangle \underline{\pi} \rangle$ for some L. Note that $L\langle \overline{t} \rangle \downarrow = \overline{t}\downarrow_L$ is a fireball, indeed if $\overline{t}$ is a symbol it is given by Lemma 19 and if it as a variable it is given by Lemma 5.3. Moreover, by Lemma 5.3, $\underline{\pi}\downarrow_L$ has the form $\langle \cdot \rangle f_1 \dots f_n$. Thus, by Lemma 30.1 and the definition of fireballs, $\langle \overline{t} \rangle \underline{\pi}\downarrow_L$ is a fireball too. Therefore by Lemma 35, $L\langle \langle \overline{t} \rangle \underline{\pi} \rangle$ is normal. ■

Proof of Theorem 9 (page 12)

Proof: the theorem follows from Lemma 38, Lemma 39 and Lemma 40. ■

Proof of Theorem 10 (page 12)

Proof: the proof follows from Theorem 3 applied to Theorem 8, and Theorem 6 applied to Theorem 9 and Corollary 1. Bi-linearity of the machine requires to show that the commutative steps are implementable in constant time, while the principal ones in time $O(|t|)$. The machine is meant to be implemented using a representation of codes using pointers, in particular for variables, so that the environment can be accessed in constant time. Assuming this, all rules except the exponential one evidently take constant time on a RAM machine, because they amount to moving pointers. The exponential rule requires $O(|t|)$ because it copies and α-renames a value v. Both these operations take time $O(|v|)$. The value invariant (Lemma 5.2) guarantees $|v| \leq |t|$. Additional considerations on the cost of similar rules can be found in [15] (page 9 and 11, paragraphs *Abstract Considerations on Concrete Implementations*). ■

APPENDIX H
PROOFS OMITTED FROM SECT. XIII
(OPTIMISING USEFUL REDUCTION: UNCHAINING FBC AND THE UNCHAINING GLAMOUR)

We prove Lemma 42 first; then we address Theorem 12 ($(\rightarrow_{\mathtt{f}}, \multimap_{\mathtt{of}})$ is a globally bounded high-level implementation system) and Proposition 5 ($\equiv$ is a Strong Bisimulation).

For chain-starting contexts $\overleftarrow{C}^x$, we need prove that their hole is indeed the left end of the chain, with the help of a preliminary lemma.

Lemma 41. *Let $I\langle x \rangle$ s.t. I does not capture x. Then $I\langle x \rangle \downarrow = x$.*

Proof: by induction on I. Cases:

1) *Base* $I = \langle \cdot \rangle$. Then $I\langle x \rangle \downarrow = x\downarrow = x$.
2) *Inductive* $I = I\langle y \rangle [y \leftarrow I']$. Then
$$I\langle x \rangle \downarrow = I\langle y \rangle [y \leftarrow I'\langle x \rangle] \downarrow = I\langle y \rangle \downarrow \{y \leftarrow I'\langle x \rangle \downarrow\} =_{\text{i.h.}} y\{y \leftarrow I'\langle x \rangle \downarrow\} =_{\text{i.h.}} y\{y \leftarrow x\} = x$$
3) *Closure* $I = I[y \leftarrow t]$. Then
$$I\langle x \rangle \downarrow = I\langle x \rangle [y \leftarrow t] \downarrow = I\langle x \rangle \downarrow \{y \leftarrow t\downarrow\} =_{\text{i.h.}} x\{y \leftarrow t\downarrow\} = x$$
■

Lemma 42. *Let $C\langle \underline{x} \rangle$ s.t. C does not capture x. Then there exists y s.t. $C\langle x \rangle = \overleftarrow{C}^x \langle y \rangle$ and $y\downarrow_{\overleftarrow{C}^x} = x$.*

Proof: by induction on C. Cases:

1) *Base, i.e.* $C = S\langle y \rangle [y \leftarrow I]$. Then
$$C\langle x \rangle = S\langle y \rangle [y \leftarrow I\langle x \rangle] = \overleftarrow{S'\langle y \rangle [y \leftarrow I]}^x \langle y \rangle$$
Now, $y\downarrow_{\overleftarrow{C}^x} = y\downarrow_{S[y \leftarrow I\langle x \rangle]} = I\langle x \rangle \downarrow =_{L.41} x$
2) *Inductive, i.e.* $C = C'\langle z \rangle [z \leftarrow I]$. Then
$$\begin{array}{ll} C\langle x \rangle & = \\ C'\langle z \rangle [z \leftarrow I\langle x \rangle] & =_{\text{i.h.}} \\ \overleftarrow{C'}^z \langle y \rangle [z \leftarrow I\langle x \rangle] & = \\ \overleftarrow{C'\langle z \rangle [z \leftarrow I]}^x \langle y \rangle & \end{array}$$
Now,
$$\begin{array}{ll} y\downarrow_{\overleftarrow{C}^x} & = \\ y\downarrow_{\overleftarrow{C'}^z_{[z \leftarrow I\langle x \rangle]}} & = \\ y\downarrow_{\overleftarrow{C'}^z} \{z \leftarrow I\langle x \rangle \downarrow\} & =_{\text{i.h.}} \\ z\{z \leftarrow I\langle x \rangle \downarrow\} = I\langle x \rangle \downarrow =_{L.41} x & \end{array}$$
3) *Closure, i.e.* $C = S'\langle C' \rangle$. Then
$$C\langle x \rangle = S'\langle C'\langle x \rangle \rangle =_{\text{i.h.}} S'\langle \overleftarrow{C}^x \langle y \rangle \rangle = \overleftarrow{S'\langle C \rangle}^x \langle y \rangle$$
Now,
$$y\downarrow_{\overleftarrow{C}^x} = y\downarrow_{S'\langle \overleftarrow{C}^x \rangle} =_{L.30.5} y\downarrow_{\overleftarrow{C}^x} \downarrow_{S'} =_{\text{i.h.}} x\downarrow_{S'} = x$$
because C, and thus S', does not capture x. ■

Proof of Lemma 7 (page 13)

Proof: by induction over the length of the derivation. A simple inspection of the rewriting rules shows that all values in the result of a reduction step are copies of values in the term being reduced. ■

Proof of Lemma 8 (page 13)

Proof: using the sub-term property (Lemma 7). ■

From now on we follow closely the same approach used for the Explicit FBC (Appendix B and Appendix C) and the Useful FBC (Appendix G), without the need to define proper terms first: we start characterizing normal forms; then we prove projection and we obtain Theorem 12 ($(\rightarrow_{\mathtt{f}}, \multimap_{\mathtt{of}})$ is a globally bounded high-level implementation system) as a corollary.

Lemma 43 (Normal Form Characterization)**.** *Let $t = S\langle u \rangle$ be a proper term s.t. u is $\multimap_{\mathtt{of}}$-normal and S is evaluable.*

1) *either* $u\downarrow_S$ *is a fireball,*
2) *or* $u\downarrow_S \to_{\mathtt{f}}$*, more precisely exists* S' *s.t.*
   a) $u = S'\langle x\rangle$ *with*
   b) $x\downarrow_{S'} = y$,
   c) $y \in \mathtt{fv}(u)$,
   d) $S\langle S'\rangle$ *evaluable,*
   e) $y\downarrow_S = \lambda y.w$*, and*
   f) S' *is applicative.*

Moreover, the context S' *in Point 2 is unique.*

Proof: first of all, let us show that conditions *2.a-f* imply $u\downarrow_S \multimap$. Indeed,

$$\begin{array}{ll} u\downarrow_S & =_a \\ S'\langle x\rangle\downarrow_S & =_{c\&L.30.6} \\ S'\downarrow_S\langle x\downarrow_{S\langle S'\rangle}\rangle & =_{d\&L.31.1} \\ F\langle x\downarrow_{S\langle S'\rangle}\rangle & =_b \\ F\langle y\downarrow_S\rangle & =_e \\ F\langle\lambda y.w\rangle & \end{array}$$

and $F := S'\downarrow_S$ is applicative, by d and L.30.4.

Now, we show that 1 and 2 are mutually exclusive. By Lemma 15, an unfolded term which is a fireball is $\to_{\mathtt{f}}$-normal. Then if 1 hold then 2 does not, and if 2 holds 1 does not. Therefore, in the following proof we only prove that 1 or 2 holds.

By induction on u. Cases:

1) *Variable* x. Since t is proper, $u\downarrow_S$ is a fireball.
2) *Symbol* and *Abstraction*. Note that by properness, the abstraction is an ordinary λ-term, *i.e.* it does not contain ES. Then in both cases we can apply Lemma 19, obtaining that $u\downarrow_S$ is a fireball.
3) *Application* $u = wr$. Since r is normal and $S\langle w\langle\cdot\rangle\rangle$ is an evaluable context, we can apply the *i.h.* to r, ending in one of the following two cases:
   a) *1 holds for* r. Then $S\langle\langle\cdot\rangle r\rangle$ is an evaluable context and we can apply the *i.h.* to w and obtain two cases:
      i) *1 holds for* w. Two cases:
         A) $w\downarrow_{S\langle\langle\cdot\rangle r\rangle} = A$. Then $u\downarrow_S$ is a inert, *i.e.* a fireball.
         B) $w\downarrow_{S\langle\langle\cdot\rangle r\rangle} = \lambda y.q$. Note that w cannot be itself an abstraction, otherwise u would not be normal. Then $w = L\langle y\rangle$. Now, $y\downarrow_L$ cannot be an abstraction, otherwise—again—u would not be normal. Then $y\downarrow_L = x$ for some $x \in \mathtt{fv}(w)$ (possibly $x = y$). Note that $S' := Lr$ is applicative and satisfies the other points of 2. For c, in particular, we only have to show that the content of every substitution in L unfolds to a fireball in its context (by Lemma 4.2). Note that, since t is proper, there is an evaluable context containing all the ES in t, *i.e.* the content of *every* substitution in t unfolds in its context to a fireball.
      ii) *2 holds for* w. Then 2 holds for u by taking $S' := S''r$ where S'' is the context given by the *i.h.*, as all the conditions for S' follows from those for S''. Unicity follows from the *i.h.* and the fact that no other such context can have its hole in r, because 2 does not hold for it.
   b) *2 holds for* r. Then 2 holds for u by taking $S' := wS''$ where S'' is the context given by the *i.h.*, as all the conditions for S' follows from those for S''. Unicity follows from the *i.h.* and the fact that no other such context can have its hole in w, because 1 does not hold for r.
4) *Substitution* $u = w[z\leftarrow r]$. Then $S\langle\langle\cdot\rangle[z\leftarrow r]\rangle\downarrow$ is an evaluation context and we can apply the *i.h.* to w. Two cases:
   a) *1 holds for* w. Note that since $w\downarrow_{S\langle\langle\cdot\rangle[z\leftarrow r]\rangle} =_{L.30.6} w[z\leftarrow r]\downarrow_S = u\downarrow_S$, then 1 holds for u.
   b) *2 holds for* w. Let $y \in \mathtt{fv}(w)$ be the variable and S' be the context given by the *i.h.*. Then we have two cases:
      i) $y = z$. Necessarily, r has the form $L\langle x'\rangle$ with $x'\downarrow_L = y'$, otherwise u would not be $\multimap_{\mathtt{uf}}$-normal. Taking $S'' := S'[z\leftarrow r]$ it is easily seen that 2 holds for u with respect to x and y'. Unicity follows from the *i.h.*.
      ii) $y \neq z$. Taking $S'' := S'[z\leftarrow r]$ it is easily seen that 2 holds for u with respect to x and y. Unicity follows from the *i.h.*. ■

Corollary 7 (Normal Forms Unfold to Normal Forms)**.** *Let* t *be a closed proper term. If* t *is* $\multimap_{\mathtt{of}}$*-normal then* $t\downarrow$ *is* $\to_{\mathtt{f}}$*-normal.*

Proof: note that taking $S := \langle\cdot\rangle$ and $u := t$ and applying Lemma 43 one obtains that $t\downarrow$ is a fireball, because the second case cannot happen, given that u now is closed. By Lemma 15, $t\downarrow$ is $\to_{\mathtt{f}}$-normal. ■

To prove the projection lemma we need to prove first as a technical lemma another sufficient condition for a context to be evaluable. The condition is based on the definition of position of a redex.

The *position* of a $\multimap_{\mathtt{om}}$-redex is S. The position of $\multimap_{\mathtt{oec}}$ and $\multimap_{\mathtt{oes}}$ redexes is the application that makes applicative the evaluable context in the side condition. Note that the position of a redex is always a context exposing an application constructor.

Lemma 44 (Projection)**.** *Let* $t = S\langle u\rangle \multimap_{\mathtt{of}} S'\langle w\rangle = r$ *by reducing a redex whose position lies in* u*. If the redex is*

1) Multiplicative*: then* $u\downarrow_S \to_{\mathtt{f}} w\downarrow_{S'}$ *and* $t\downarrow \to_{\mathtt{f}} r\downarrow$*;*
2) Shallow or Chain Exponential*: then* $u\downarrow_S \to_{\mathtt{f}}$ *and* $t\downarrow = r\downarrow\to_{\mathtt{f}}$.

In both cases $u\downarrow_S$ *is not a fireball.*

Proof: the fact that in both cases $u\downarrow_S$ is not a fireball, follows from Lemma 15 and the fact that $u\downarrow_S$ reduces. Cases:

1) *Multiplicative*. Exactly as in the proof of Lemma 35.
2) *Exponential*. We take $t\downarrow = r\downarrow$ for granted, because a substitution step by definition does not change the unfolding. Similarly to the previous case, $t\downarrow \to_{\mathtt{f}}$ follows from $u\downarrow_S \to_{\mathtt{f}}$. Indeed, by Lemma 31.1, S is evaluable, and by Lemma 4.1 $S\downarrow$ is an evaluation context, so:

$$t\downarrow = S\langle u\rangle\downarrow =_{L.30.6} S\downarrow\langle u\downarrow_S\rangle \to_{\mathtt{f}}$$

Now we prove $u\downarrow \to_{\mathtt{f}}$. We have $u = S'\langle L\langle x\rangle r\rangle$ and $t = S\langle S'\langle L\langle x\rangle r\rangle\rangle$. Let us show that for both exponential redexes x unfolds to an abstraction. In t there is somewhere (in L, S', or S) a substitution $[x\leftarrow q]$. Now, if $q = L'\langle v\rangle$ then we have a $\multimap_{\mathtt{oes}}$-redex (because v is an abstraction). If instead $q = L'\langle y\rangle$ then we have a $\multimap_{\mathtt{oec}}$-redex and t writes also as $S''\langle C\langle y\rangle[y\leftarrow L''\langle v\rangle]\rangle$ with $C\langle y\rangle = \overleftarrow{C}^y\langle x\rangle$, $x\downarrow_{\overleftarrow{C}^y} = y$ (by Lemma 42), and s.t. the two contexts $S\langle S'\langle Lr\rangle\rangle$ and $S''\langle\overleftarrow{C}^y[y\leftarrow L''\langle v\rangle]\rangle$ coincide. Then

$$\begin{array}{lcl}
x\downarrow_{S\langle S'\langle Lr\rangle\rangle} & = & \\
x\downarrow_{S''\langle\overleftarrow{C}^y[y\leftarrow L''\langle v\rangle]\rangle} & = & \text{(Lemma 30.5)}\\
x\downarrow_{\overleftarrow{C}^y[y\leftarrow L''\langle v\rangle]}\downarrow_{S''} & = & \\
x\downarrow_{\overleftarrow{C}^y}\{y\leftarrow L''\langle v\rangle\downarrow\}\downarrow_{S''} & = & (x\downarrow_{\overleftarrow{C}^y} = y)\\
y\{y\leftarrow L''\langle v\rangle\downarrow\}\downarrow_{S''} & = & (L''\langle v\rangle\downarrow \text{ is an abst.})\\
y\{y\leftarrow v'\}\downarrow_{S''} & = & \\
v'\downarrow_{S''} & = & (v'\downarrow_{S''} \text{ is an abst.})\\
v''
\end{array}$$

Summing up, $x\downarrow_{S\langle S'\langle L\rangle\rangle} = \lambda y.w$. Moreover, by hypothesis $S\langle S'\rangle$ is evaluable, and so by Lemma 31.1 $S'\downarrow_S$ is an evaluation context. Finally, $r\downarrow_{S\langle S'\rangle}$ is a fireball, because $S\langle S'\langle\langle\cdot\rangle r\rangle\rangle$ is evaluable. Then

$$\begin{array}{lcl}
u\downarrow_S & = & \\
S'\langle L\langle x\rangle r\rangle\downarrow_S & = & \text{(Lemma 30.6)}\\
S'\downarrow_S\langle(L\langle x\rangle r)\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.1)}\\
S'\downarrow_S\langle L\langle x\rangle\downarrow_{S\langle S'\rangle} r\downarrow_{S\langle S'\rangle}\rangle & = & \text{(Lemma 30.6)}\\
S'\downarrow_S\langle x\downarrow_{S\langle S'\langle L\rangle\rangle} r\downarrow_{S\langle S'\rangle}\rangle & = & (x\downarrow_{S\langle S'\langle L\rangle\rangle} = \lambda y.w)\\
S'\downarrow_S\langle(\lambda y.w)r\downarrow_{S\langle S'\rangle}\rangle & \to_{\mathtt{f}} & (S'\downarrow_S \text{ is an ev. cont.}\\
 & & \ \& \ r\downarrow_{S\langle S'\rangle} \text{ a fireball})\\
S'\downarrow_S\langle w[y\leftarrow r\downarrow_{S\langle S'\rangle}]\rangle & &
\end{array}$$

■

Lemma 45 (Positional Determinism)**.** *Let t be a term and $S\langle S_1\rangle$ and $S\langle S_2\rangle$ positions of $\multimap_{\mathtt{of}}$-redexes. Then $S_1 = S_2$.*

Proof: by induction on S_1. Cases:

1) *Empty* $S_1 = \langle\cdot\rangle$. Cases:
   a) *Multiplicative Redex, i.e.* $u = L\langle\lambda x.r\rangle q$ with $q\downarrow_S$ a fireball. Now, S_2 cannot lie in $L\langle\lambda x.r\rangle$, otherwise by Lemma 35 $L\langle\lambda x.r\rangle\downarrow_{S\langle S_2\rangle}$ would not be a fireball, while by Lemma 19 it does. Nor S_2 can lie in q, otherwise again by Lemma 35 $q\downarrow_S$ would not be a fireball. Then necessarily $S_2 = S_1 = \langle\cdot\rangle$.
   b) *Exponential Redex, i.e.* $u = S'\langle L\langle x\rangle r\rangle$. Now, S_2 cannot lie in $L\langle x\rangle$, otherwise by Lemma 35 $L\langle x\rangle\downarrow_{S\langle S_2\rangle}$ would not be a fireball, while by the hypothesis on the $\multimap_{\mathtt{e}}$-redex it does (it is an abstraction). Nor S_2 can lie in r, otherwise again by Lemma 35 $r\downarrow_S$ would not be a fireball, while by the hypothesis on the $\multimap_{\mathtt{e}}$-redex it does. Then necessarily $S_2 = S_1 = \langle\cdot\rangle$.
2) *Right Application* $S_1 = rS_1'$ and $t = rS_1'\langle q\rangle$. By Lemma 35, $S_1'\langle q\rangle\downarrow_{S\langle r\langle\cdot\rangle\rangle}$ has a $\to_{\mathtt{f}}$-redex and it is not a fireball, so no redexes can lie to its left, in particular S_2 does not lie in r. By Lemma 35, $S_1'\langle q\rangle\downarrow_{S\langle r\langle\cdot\rangle\rangle}$ is not a fireball, and so S_2 cannot be empty (*i.e.* $rS_1'\langle q\rangle$ cannot be the position of a $\multimap_{\mathtt{m}}$-redex). Then, $S_2 = uS_2'$, and the statement follows from the *i.h.* applied to S_1' and S_2'.
3) *Left Application* $S_1 = S_1'q$ and $t = S_1'\langle r\rangle q$. Note that S_2 cannot lie in q, otherwise by Lemma 35 $q\downarrow_{S\langle S_1'\langle r\rangle\langle\cdot\rangle\rangle}$ has a $\to_{\mathtt{f}}$-redex and it is not a fireball, and so no redexes—in particular the one of position $S\langle S_1\rangle$—can lie to its left, absurd. And S_2 cannot be empty (*i.e.* the position of a $\multimap_{\mathtt{m}}$-redex), because then $S_1'\langle r\rangle$ would have the form $L\langle\lambda x.p\rangle$, which by Lemma 35 cannot contain the position of a redex, because by Lemma 19 $L\langle\lambda x.p\rangle\downarrow_{S\langle\langle\cdot\rangle q\rangle}$ is a fireball. Then, $S_2 = S_2'w$, and the statement follows from the *i.h.* applied to S_1' and S_2'.
4) *Substitution* $S_1 = S_1'[x\leftarrow w]$. Then necessarily $S_2 = S_2'[x\leftarrow w]$ (remember the position of a $\multimap_{\mathtt{e}}$-redex is an application) and the statement follows from the *i.h.*. ■

Note that we did not yet prove determinism, as two redexes may a priori have the same position.

Lemma 46 (Redexes Have Different Positions)**.** *Any two $\multimap_{\mathtt{of}}$-redexes in a term t have different positions.*

Proof: It is obvious that different multiplicative redexes have different positions, and that multiplicative and exponential redexes cannot have the same position. Now consider an exponential position $S\langle L\langle x\rangle t\rangle$ and let $[x\leftarrow u]$ be the substitution on x lying somewhere in S or L. If t has the form $L'\langle v\rangle$ then there is a $\multimap_{\mathtt{oes}}$ redex, and obviously there cannot be other $\multimap_{\mathtt{oes}}$ or $\multimap_{\mathtt{oec}}$ redex with the same position. If instead t has the form $L'\langle y\rangle$ then we start following the chain of substitutions leading to the abstraction. Note that there is no choice about the chain, so there can only by one $\multimap_{\mathtt{oec}}$-redex with that position. ■

Corollary 8 (Determinism)**.** *$\multimap_{\mathtt{of}}$ is deterministic.*

Proof: it follows from Lemma 45 and Lemma 46. ■

Proof of Theorem 12 (page 13)

Proof: the pair $(\to_{\mathtt{f}}, \multimap_{\mathtt{of}})$ is an high-level implementation system because of Lemma 45, Lemma 44 and Corollary 7. It is also globally bounded because we already proved the global linear bound on exponential steps (Theorem 11). ■

Proof of Proposition 5 (page 13)

Proof: omitted. All postponement proofs are similar and lengthy. In Subsect. B-B of the Appendix we proved the lemma for the Explicit FBC. Other examples can be found in the long version of [15]. ∎

APPENDIX I
PROOFS OMITTED FROM SECT. XIV
(UNCHAINING GLAMOUR)

The aim of this section is to prove Theorem 13 ((Unchaining GLAMoUr, $\multimap_{\mathtt{of}}, \equiv, \underline{\cdot}$) is a reflective explicit distillery) and the final result of the paper, Theorem 14 (the useful implementation has bilinear low level and bilinear high level complexity).

We follow closely the methodology of Appendix XII. The first step is proving that the invariants of the machine holds.

Lemma 47. $y^{H:x} = x^H$

Proof: by induction over H. ∎

Lemma 48 (Contextual Decoding)**.** *$\underline{E}$ is a substitution context; $\underline{D}$ and $\underline{\pi}$ are shallow contexts without ES.*

Proof: by induction over E, D and π. ∎

Remark 1. *if $H : x$ is compatible with E, then also H is compatible with E.*

Proof of Lemma 11 (page 14)

Proof: by induction over the length of the execution. The base case holds because $\overline{t}$ is initial. The inductive step is by cases over the kind of transition. All the verifications are trivial. Point 4 is proved as in the useful case (see Lemma 5, page 12). ∎

Proof of Lemma 12 (page 14)

Proof: the first point is trivial, we prove the other two. By induction on the length k of H. Cases:

- H is empty, *i.e.* $H = \epsilon$. By Lemma 11.6 we have $E := E_1[y{\leftarrow}x^v]E_2$. Let also $S := \langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E}$. We have $\underline{s} = \langle \underline{D}\langle\langle y^{\epsilon:y}\rangle\underline{\pi}\rangle\rangle\underline{E} = S\langle y^{\epsilon:y}\rangle$ and
  1) $L_s = \underline{E_2}$ and $C_s = \langle \underline{D}\langle\langle y\rangle\underline{\pi}\rangle\rangle\underline{E_1}[y{\leftarrow}\langle\cdot\rangle]$, that (by Lemma 48) has the form $S\langle y\rangle[y{\leftarrow}I]$, and so it is a chain context,
  2) Now,

$$\begin{array}{ll} \underline{s} & = \\ S_s\langle y^{\epsilon:y}\rangle & = \\ \langle \underline{D}\langle\langle y^{\epsilon:y}\rangle\underline{\pi}\rangle\rangle\underline{E} & = \\ \langle \underline{D}\langle\langle y\rangle\underline{\pi}\rangle\rangle\underline{E} & = \\ \langle \underline{D}\langle\langle y\rangle\underline{\pi}\rangle\rangle\underline{E_1[y{\leftarrow}x^v]E_2} & = \\ \langle\langle \underline{D}\langle\langle y\rangle\underline{\pi}\rangle\rangle\underline{E_1[y{\leftarrow}x]}\rangle\underline{E_2} & = \\ L_s\langle\langle \underline{D}\langle\langle y\rangle\underline{\pi}\rangle\rangle\underline{E_1}[y{\leftarrow}x]\rangle & = \\ L_s\langle C_s\langle x\rangle\rangle & \end{array}$$

  and

$$\begin{array}{ll} L_s\langle \overleftarrow{C_s}^x\rangle & = \\ L_s\langle \overleftarrow{\langle \underline{D}\langle\langle y\rangle\underline{\pi}\rangle\rangle\underline{E_1}[y{\leftarrow}\langle\cdot\rangle]}^x\rangle & = \\ L_s\langle\langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E_1}[y{\leftarrow}x]\rangle & = \\ \langle\langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E_1}[y{\leftarrow}x]\rangle\underline{E_2} & = \\ \langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E_1[y{\leftarrow}x^v]E_2} & = \\ \langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E} & = \\ S_s & \end{array}$$

- Non-empty, *i.e.* $H = H' : z$. By Lemma 11.6 we have $E = E_1[z{\leftarrow}y^v]E_2[y{\leftarrow}x^v]E_3$ and $S := \langle \underline{D}\langle\underline{\pi}\rangle\rangle\underline{E}$, so that $\underline{s} = \langle \underline{D}\langle\langle y^{H':z:y}\rangle\underline{\pi}\rangle\rangle\underline{E} = S\langle y^{H':z:y}\rangle$. Note that by Remark 1 we can apply the *i.h.* to the state $s' = (D, H' : z, y, \pi, E)$, and we will do it in the following points.
  Now,
  1) $L_s = \underline{E_3}$ and for C_s, note that we have

$$C_{s'} = \langle \underline{D}\langle\langle y^{H':z}\rangle\underline{\pi}\rangle\rangle\underline{E_1}[z{\leftarrow}\langle\cdot\rangle]$$

  and that by *i.h.* $C_{s'}$ is a chain context. Then

$$\begin{array}{ll} C_s & = \\ \langle \underline{D}\langle\langle y^{H':z}\rangle\underline{\pi}\rangle\rangle\underline{E_1[z{\leftarrow}y^v]E_2}[y{\leftarrow}\langle\cdot\rangle] & = \\ \langle\langle \underline{D}\langle\langle y^{H':z}\rangle\underline{\pi}\rangle\rangle\underline{E_1}[z{\leftarrow}y]\rangle\underline{E_2}[y{\leftarrow}\langle\cdot\rangle] & = \\ \langle C_{s'}\langle y\rangle\rangle\underline{E_2}[y{\leftarrow}\langle\cdot\rangle] & \end{array}$$

  and so C_s is a chain context.
  2) Note that $L_{s'} = \underline{E_2[y{\leftarrow}x^v]E_3}$, and so

$$\begin{array}{ll} L_s\langle C_s\langle x\rangle\rangle & = \\ \langle\langle C_{s'}\langle y\rangle\rangle\underline{E_2}[y{\leftarrow}x]\rangle\underline{E_3} & = \\ \langle C_{s'}\langle y\rangle\rangle\underline{E_2[y{\leftarrow}x^v]E_3} & = \\ L_{s'}\langle C_{s'}\langle y\rangle\rangle & =_{\textit{i.h.}} \\ \underline{s} & \end{array}$$

  Then note that

$$\begin{array}{ll} \overleftarrow{C_s}^x & = \\ \overleftarrow{\langle C_{s'}\langle y\rangle\rangle\underline{E_2}[y{\leftarrow}\langle\cdot\rangle]}^x & = \\ \langle \overleftarrow{C_{s'}\langle y\rangle}\rangle\underline{E_2}^y[y{\leftarrow}x] & = \\ \langle \overleftarrow{C_{s'}}^y\rangle\underline{E_2}[y{\leftarrow}x] & \end{array}$$

  Now we conclude with

$$\begin{array}{ll} L_s\langle \overleftarrow{C_s}^x\rangle & = \\ \langle \overleftarrow{C_s}^x\rangle\underline{E_3} & = \\ \langle\langle \overleftarrow{C_{s'}}^y\rangle\underline{E_2}[y{\leftarrow}x]\rangle\underline{E_3} & = \\ \langle \overleftarrow{C_{s'}}^y\rangle\underline{E_2[y{\leftarrow}x^v]E_3} & = \\ L_{s'}\langle \overleftarrow{C_{s'}}^y\rangle & =_{\textit{i.h.}} \\ S_{s'} & = \\ S_s & \end{array}$$

∎

Lemma 49 (Unchaining GLAMoUr Distillation)**.** *Let s be a reachable state. Then:*

1) Commutative*: if $s \rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}} s'$ then $\underline{s} = \underline{s'}$;*
2) Multiplicative*: if $s \rightsquigarrow_{\mathtt{um}} s'$ then $\underline{s} \multimap_{\mathtt{um}} \equiv \underline{s'}$;*
3) Shallow Exponential*: if $s \rightsquigarrow_{\mathtt{oes}} s'$ then $\underline{s} \multimap_{\mathtt{oes}} \underline{s'}$;*
4) Chain Exponential*: if $s \rightsquigarrow_{\mathtt{oec}} s'$ then $\underline{s} \multimap_{\mathtt{oec}} \underline{s'}$.*

Proof: we list the transition in the order they appear in the definition of the machine.

- Case $(D, \epsilon, \overline{t}\overline{u}, \pi, E) \rightsquigarrow_{\mathtt{c}_1} (D : (\overline{t}, \pi), \epsilon, \overline{u}, \epsilon, E)$:

$$\begin{array}{ll}
\underline{(D, \epsilon, \overline{t}\overline{u}, \pi, E)} & = \\
\underline{\langle \underline{D} \langle \langle (\overline{t}\overline{u})^{\epsilon} \rangle \underline{\pi} \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t}\overline{u} \rangle \underline{\pi} \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t} \langle \overline{u} \rangle \rangle \underline{\pi} \rangle \rangle \underline{E}} & = \\
\underline{\langle D : (\overline{t}, \pi) \langle \overline{u} \rangle \rangle \underline{E}} & = \\
\underline{\langle \overline{D : (\overline{t}, \pi)} \langle \langle \overline{u} \rangle \underline{\epsilon} \rangle \rangle \underline{E}} & = \\
\underline{\langle \overline{D : (\overline{t}, \pi)} \langle \langle \overline{u^{\epsilon}} \rangle \underline{\epsilon} \rangle \rangle \underline{E}} & = \\
\underline{(\overline{D : (\overline{t}, \pi)}, \epsilon, \overline{u}, \epsilon, E)} &
\end{array}$$

- Case $(D, \epsilon, \lambda x.\overline{t}, \phi^l : \pi, E) \rightsquigarrow_{\mathtt{om}} (D, \epsilon, \overline{t}, \pi, E[x \leftarrow \phi^l])$:

$$\begin{array}{ll}
\underline{(D, \epsilon, \lambda x.\overline{t}, \phi^l : \pi, E)} & = \\
\underline{\langle \underline{D} \langle \langle \lambda x.\overline{t}^{\epsilon} \rangle \phi^l : \pi \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \lambda x.\overline{t} \rangle \phi^l : \pi \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle (\lambda x.\overline{t}) \phi \rangle \underline{\pi} \rangle \rangle \underline{E}} & \multimap_{\mathtt{um}} \\
\underline{\langle \underline{D} \langle \langle \overline{t}[x \leftarrow \underline{\phi}] \rangle \underline{\pi} \rangle \rangle \underline{E}} & \equiv \\
\underline{\langle \underline{D} \langle \langle \overline{t} \rangle \underline{\pi} \rangle [x \leftarrow \phi] \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t} \rangle \underline{\pi} \rangle \rangle \underline{E[x \leftarrow \phi^l]}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t^{\epsilon}} \rangle \underline{\pi} \rangle \rangle \overline{E[x \leftarrow \phi^l]}} & = \\
\underline{(D, \epsilon, \overline{t}, \pi, \overline{E[x \leftarrow \phi^l]})} &
\end{array}$$

  The multiplicative step is justified by Lemma 11.4 and Lemma 11.3. The bisimulation step is justified by Lemma 24.

- Case

$$(D : (\overline{t}, \pi), \epsilon, \lambda x.\overline{u}, \epsilon, E) \rightsquigarrow_{\mathtt{c}_2} (D, \epsilon, \overline{t}, (\lambda x.\overline{u})^v : \pi, E)$$

  We have

$$\begin{array}{ll}
\underline{(D : (\overline{t}, \pi), \epsilon, \lambda x.\overline{u}, \epsilon, E)} & = \\
\underline{\langle D : (\overline{t}, \pi) \langle \langle \lambda x.\overline{u}^{\epsilon} \rangle \underline{\epsilon} \rangle \rangle \underline{E}} & = \\
\underline{\langle \overline{D : (\overline{t}, \pi)} \langle \langle \lambda x.\overline{u} \rangle \underline{\epsilon} \rangle \rangle \underline{E}} & = \\
\underline{\langle \overline{D : (\overline{t}, \pi)} \langle \lambda x.\overline{u} \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t} (\lambda x.\overline{u}) \rangle \underline{\pi} \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t} \rangle (\lambda x.\overline{u})^v : \pi \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t}^{\epsilon} \rangle \overline{(\lambda x.\overline{u})^v : \pi} \rangle \rangle \underline{E}} & = \\
\underline{(D, \epsilon, \overline{t}, \overline{(\lambda x.\overline{u})^v : \pi}, E)} &
\end{array}$$

- Case

$$(D : (\overline{t}, \pi), \epsilon, a, \pi', E) \rightsquigarrow_{\mathtt{c}_3} (D, \epsilon, \overline{t}, (a, \pi')^A : \pi, E)$$

  We have

$$\begin{array}{ll}
\underline{(D : (\overline{t}, \pi), \epsilon, a, \pi', E)} & = \\
\underline{\langle D : (\overline{t}, \pi) \langle \langle \underline{a}^{\epsilon} \rangle \underline{\pi'} \rangle \rangle \underline{E}} & = \\
\underline{\langle \overline{D : (\overline{t}, \pi)} \langle \langle \underline{a} \rangle \underline{\pi'} \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t} \langle a \rangle \underline{\pi'} \rangle \underline{\pi} \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t} \rangle (a, \pi')^A : \pi \rangle \rangle \underline{E}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{t}^{\epsilon} \rangle \overline{(a, \pi')^A : \pi} \rangle \rangle \underline{E}} & = \\
\underline{(D, \epsilon, \overline{t}, \overline{(a, \pi')^A : \pi}, E)} &
\end{array}$$

- Case

$$\begin{array}{l}
(D : (\overline{t}, \pi), \epsilon, x, \pi', E_1[x \leftarrow \phi^A] E_2) \rightsquigarrow_{\mathtt{c}_4} \\
(D, \epsilon, \overline{t}, (x, \pi')^A : \pi, E_1[x \leftarrow \phi^A] E_2)
\end{array}$$

  We have

$$\begin{array}{ll}
\underline{(D : (\overline{t}, \pi), \epsilon, x, \pi', E_1[x \leftarrow \phi^A] E_2)} & = \\
\ldots & \\
\underline{(D, \epsilon, \overline{t}, (x, \pi')^A : \pi, E_1[x \leftarrow \phi^A] E_2)} &
\end{array}$$

  The proof is the one for the previous case $\rightsquigarrow_{\mathtt{c}_3}$, by replacing a with x and instantiating E with $E_1[x \leftarrow \phi^A] E_2$.

- Case

$$\begin{array}{l}
(D : (\overline{t}, \pi), \epsilon, x, \epsilon, E_1[x \leftarrow \overline{u}^v] E_2) \rightsquigarrow_{\mathtt{c}_5} \\
(D, \epsilon, \overline{t}, x^v : \pi, E_1[x \leftarrow \overline{u}^v] E_2)
\end{array}$$

  We have

$$\begin{array}{ll}
\underline{(D : (\overline{t}, \pi), \epsilon, x, \epsilon, E_1[x \leftarrow \overline{u}^v] E_2)} & = \\
\ldots & = \\
\underline{(D, \epsilon, \overline{t}, x^v : \pi, E_1[x \leftarrow \overline{u}^v] E_2)} &
\end{array}$$

  The proof is the one for the previous case $\rightsquigarrow_{\mathtt{c}_4}$, by replacing $(\lambda x.\overline{u})$ with x and instantiating E with $E_1[x \leftarrow \overline{u}^v] E_2$.

- Case

$$\begin{array}{l}
(D, \epsilon, x, \phi^l : \pi, E_1[x \leftarrow \overline{v}^v] E_2) \rightsquigarrow_{\mathtt{oes}} \\
(D, \epsilon, \overline{v}^{\alpha}, \phi^l : \pi, E_1[x \leftarrow \overline{v}^v] E_2)
\end{array}$$

  We have

$$\begin{array}{ll}
\underline{(D, \epsilon, x, \phi^l : \pi, E_1[x \leftarrow \overline{v}^v] E_2)} & = \\
\underline{\langle \underline{D} \langle \langle x^{\epsilon} \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E_1[x \leftarrow \overline{v}^v] E_2}} & = \\
\underline{\langle \underline{D} \langle \langle x \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E_1[x \leftarrow \overline{v}^v] E_2}} & \multimap_{\mathtt{oes}} \text{ (by L.11.4)} \\
\underline{\langle \underline{D} \langle \langle \overline{v}^{\alpha} \rangle \underline{\phi^l : \pi} \rangle \rangle \underline{E_1[x \leftarrow \overline{v}^v] E_2}} & = \\
\underline{\langle \underline{D} \langle \langle \overline{v}^{\alpha \epsilon} \rangle \underline{\phi^l : \pi} \rangle \rangle \underline{E_1[x \leftarrow \overline{v}^v] E_2}} & = \\
\underline{(D, \epsilon, \overline{v}^{\alpha}, \phi^l : \pi, \overline{E_1[x \leftarrow \overline{v}^v] E_2})} &
\end{array}$$

- Case

$$\begin{array}{l}
(D, H, x, \phi^l : \pi, E_1[x \leftarrow y^v] E_2) \rightsquigarrow_{\mathtt{c}_6} \\
(D, H : x, y, \phi^l : \pi, E_1[x \leftarrow y^v] E_2)
\end{array}$$

$$\begin{array}{ll}
\underline{(D, H, x, \phi^l : \pi, E_1[x \leftarrow y^v] E_2)} & = \\
\underline{\langle \underline{D} \langle \langle x^H \rangle \underline{\phi^l : \pi} \rangle \rangle \underline{E_1[x \leftarrow y^v] E_2}} & =_{L.47} \\
\underline{\langle \underline{D} \langle \langle y^{H:x} \rangle \underline{\phi^l : \pi} \rangle \rangle \underline{E_1[x \leftarrow y^v] E_2}} & = \\
\underline{(D, H : x, y, \phi^l : \pi, E_1[x \leftarrow y^v] E_2)} &
\end{array}$$

- Case $s' := (D, H : y, x, \phi^l : \pi, E^{\bullet}) \rightsquigarrow_{\mathtt{oec}} (D, H, y, \phi^l : \pi, E^{\circ}) = s$, where $E^{\bullet} = E_1[y \leftarrow x^v] E_2[x \leftarrow \overline{v}^v] E_3$, and $E^{\circ} = E_1[y \leftarrow \overline{v}^{\alpha v}] E_2[x \leftarrow \overline{v}^v] E_3$. Note that we have:
  1) $L_{s',y} = \underline{E_2[x \leftarrow \overline{v}^v] E_3}$
  2) $C_{s',y} = \overline{\langle \underline{D} \langle \langle y^H \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E_1}[y \leftarrow \langle \cdot \rangle]}$

  Then,

$$\begin{array}{ll}
\underline{(D, H : y, x, \phi^l : \pi, E^{\bullet})} & = \\
\underline{\langle \underline{D} \langle \langle x^{H:y} \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E^{\bullet}}} & = \\
\underline{\langle \underline{D} \langle \langle y^H \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E^{\bullet}}} & =_{L.12} \\
L_{s',y} \langle C_{s',y} \langle x \rangle \rangle & = \\
\langle C_{s',y} \langle x \rangle \rangle \underline{E_2[x \leftarrow \overline{v}^v] E_3} & = \\
\langle \langle C_{s',y} \langle x \rangle \rangle \underline{E_2[x \leftarrow \overline{v}]} \rangle \underline{E_3} & \multimap_{\mathtt{oec}} \\
\langle \langle C_{s',y} \langle \overline{v}^{\alpha} \rangle \rangle \underline{E_2[x \leftarrow \overline{v}]} \rangle \underline{E_3} & = \\
\langle \langle \underline{\langle \underline{D} \langle \langle y^H \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E_1}[y \leftarrow \overline{v}^{\alpha}]} \rangle \underline{E_2[x \leftarrow \overline{v}]} \rangle \underline{E_3} & = \\
\underline{\langle \underline{D} \langle \langle y^H \rangle \underline{\phi^v : \pi} \rangle \rangle \underline{E^{\circ}}} & = \\
\underline{(D, H, y, \phi^l : \pi, E^{\circ})} &
\end{array}$$

  The chain exponential step is justified because

1) $H : y$ is compatible with $E^\bullet$, and so we can apply L.12, obtaining $L_{s',y}\langle\overleftarrow{C_{s',y}}^{x}\rangle = \langle \underline{D\langle\phi^l : \pi\rangle}\rangle \underline{E^\bullet}$
2) Lemma 11.4 guarantees that such a context—which is the context in the side-condition of the rule— is evaluable. It is also obviously applicative (because the stack has the form $\phi^l : \pi$). ■

Lemma 50 (Determinism)**.** *The transition relation $\rightsquigarrow$ of the Unchaining GLAMoUr is deterministic.*

Proof: a simple inspection of the reduction rules shows no critical pairs. ■

Lemma 51 (Progress)**.** *if s is reachable, $\mathtt{nf}_{\mathtt{c}}(s) = s$ and $\underline{s} \multimap_{\mathtt{x}} t$ with $\mathtt{x} \in \{\mathtt{om}, \mathtt{oes}, \mathtt{oec}\}$, then there exists s' such that $s \rightsquigarrow_{\mathtt{x}} s'$, i.e., s is not final.*

Proof: by Lemma 50 and Lemma 49 it is sufficient to show that every reachable stuck state decodes to a normal form. The only stuck forms are:

- *Error states.*
  1) *Problem with the heap.* $(D, H : y, \overline{t}, \pi, E)$ when $\overline{t}$ is not a variable bound in E to a ϕ^v or π is empty or y is not bound to $\overline{t}$ in E. The state is not reachable because it would violate the invariant Lemma 11.6.
  2) *Problem with the environment.* The state is (D, H, x, π, E) where x is not defined in E or it is defined to be a $\overline{t}^v$ where $\overline{t}$ is not a variable or a value.
     The state is not reachable because it would violate either the invariant in Lemma 11.1 or the invariant in Lemma 11.3.
- *Final states.* Cases:
  1) *The result is/unfolds to a value.* The state is $(\epsilon, \epsilon, \overline{t}, \epsilon, E)$ with $\overline{t}$ an abstraction or a variable bound in E to a ϕ^v. By Lemma 37, $\underline{(\epsilon, \epsilon, \overline{t}, \epsilon, E)} = \langle\overline{t}\rangle\underline{E} = L\langle\overline{t}\rangle$ for some L. Note that $\underline{L\langle\overline{t}\rangle\downarrow = \overline{t}\downarrow_L}$ is a fireball, indeed if $\overline{t}$ is an abstraction it is given by Lemma 19 and if it as a variable it is given by Lemma 11.3. Thus by Lemma 44, $L\langle\overline{t}\rangle$ is normal.
  2) *The result is/unfolds to a inert.* The state is $(\epsilon, \epsilon, \overline{t}, \pi, E)$ with $\overline{t}$ a symbol a or a variable bound in E to a ϕ^A.
     By Lemma 37, $\underline{(\epsilon, \epsilon, \overline{t}, \pi, E)} = \langle\langle\overline{t}\rangle\underline{\pi}\rangle\underline{E} = L\langle\langle\overline{t}\rangle\underline{\pi}\rangle$ for some L. Note that $\underline{L\langle\overline{t}\rangle\downarrow} = \overline{t}\downarrow_L$ is a fireball, indeed if $\overline{t}$ is a symbol it is given by Lemma 19 and if it as a variable it is given by Lemma 11.3. Thus by Lemma 44, $L\langle\overline{t}\rangle$ is normal. ■

Proof of Theorem 13 (page 14)

Proof: the theorem follows from Lemma 49, Lemma 50 and Lemma 51. ■

A. Proofs Omitted From Subsect. XIV-A (Bilinearity: Principal vs Commutative Analysis)

In the remaining of the appendix we prove bilinearity of $\rightsquigarrow_{\mathtt{c}}$. We begin redoing the proof for $\rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}}$, that is almost identical to that of the GLAMoUr.

Lemma 52 (Size Bounded)**.** *Let $s = (D, \overline{u}, \pi, E)$ be a state reached by an execution ρ of initial code $\overline{t}$. Then $|s| \leq (1 + |\rho|_{\mathtt{oes}})|\overline{t}| - |\rho|_{c_{1-5}}$.*

Proof: the same reasoning as for the useful case (Lemma 6) provides the proof for $\rightsquigarrow_m, \rightsquigarrow_{\mathtt{oes}}, \rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}}$, while for the new transitions $\rightsquigarrow_{\mathtt{c}_6}$ and $\rightsquigarrow_{\mathtt{oec}}$ it is enough to note that they do not change the size of the state. ■

Corollary 9 (Termination and Bilinearity of $\rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}}$)**.** *Let s be a state reached by an execution ρ of initial code $\overline{t}$. Then $|\rho|_{c_{1-5}} \leq (1 + |\rho|_{\mathtt{e}})|\overline{t}| = O(|\rho|_p \cdot |\overline{t}|)$. In particular, $\rightsquigarrow_{\mathtt{c}_{1,2,3,4,5}}$ terminates.*

Proof of Corollary 3 (page 15)

Proof: combining Corollary 9 with Lemma 13. ■

Proof of Theorem 14 (page 15)

Proof: the proof follows from Theorem 3 applied to Theorem 12, and Theorem 6 applied to Theorem 13 and Corollary 3. For the implementability of the steps we refer to the proof of Theorem 10. ■