Types of Fireballs

Beniamino Accattoli, Giulio Guerrieri

Types of Fireballs

Beniamino Accattoli[1] and Giulio Guerrieri[2]

[1] Inria & LIX, École Polytechnique, UMR 7161, beniamino.accattoli@inria.fr
[2] Università di Bologna, Dipartimento di Informatica—Scienza e Ignegneria (DISI), Bologna, Italy, giulio.guerrieri@unibo.it

Abstract. The good properties of Plotkin's call-by-value lambda-calculus crucially rely on the restriction to weak evaluation and closed terms. Open call-by-value is the more general setting where evaluation is weak but terms may be open. Such an extension is delicate, and the literature contains a number of proposals. Recently, Accattoli and Guerrieri provided detailed operational and implementative studies of these proposals, showing that they are equivalent from the point of view of termination, and also at the level of time cost models.
This paper explores the denotational semantics of open call-by-value, adapting de Carvalho's analysis of call-by-name via multi types (aka non-idempotent intersection types). Our type system characterises normalisation and thus provides an adequate relational semantics. Moreover, type derivations carry quantitative information about the cost of evaluation: their size bounds the number of evaluation steps and the size of the normal form, and we also characterise derivations giving exact bounds.
The study crucially relies on a new, refined presentation of the fireball calculus, the simplest proposal for open call-by-value, that is more apt to denotational investigations.

1 Introduction

The core of functional programming languages and proof assistants is usually modelled as a variation over the λ-calculus. Even when one forgets about type systems, there are in fact many λ-calculi rather than a single λ-calculus, depending on whether evaluation is weak or strong (that is, only outside or also inside abstractions), call-by-name (CbN for short), call-by-value (CbV),[3] or call-by-need, whether terms are closed or may be open, not to speak of extensions with continuations, pattern matching, fix-points, linearity constraints, and so on.

Benchmark for λ-calculi. A natural question is *what is a good λ-calculus*? It is of course impossible to give an absolute answer, because different settings value different properties. It is nonetheless possible to collect requirements that seem desirable in order to have an abstract framework that is also useful in practice. We can isolate at least six principles to be satisfied by a good λ-calculus:

[3] In CbV, function's arguments are evaluated before being passed to the function, so β-redexes can fire only when their arguments are values, i.e. abstractions or variables.

1. *Rewriting*: there should be a small-step operational semantics having nice rewriting properties. Typically, the calculus should be non-deterministic but confluent, and a deterministic evaluation strategy should emerge naturally from some good rewriting property (factorisation / standardisation theorem, or the diamond property). The *strategy emerging from the calculus* principle guarantees that the chosen evaluation is not ad-hoc.
2. *Logic*: typed versions of the calculus should be in Curry-Howard correspondences with some proof systems, providing logical intuitions and guiding principles for the features of the calculus and the study of its properties.
3. *Implementation*: there should be a good understanding of how to decompose evaluation in micro-steps, that is, at the level of abstract machines, in order to guide the design of languages or proof assistants based on the calculus.
4. *Cost model*: the number of steps of the deterministic evaluation strategy should be a reasonable time cost model,[4] so that cost analyses of λ-terms are possible, and independent of implementative choices.
5. *Denotations*: there should be denotational semantics, that is, syntax-free mathematical interpretations of the calculus that are invariant by evaluation and that reflect some of its properties. Well-behaved denotations guarantee that the calculus is somewhat independent from its own syntax, which is a further guarantee that it is not ad-hoc.
6. *Equality*: contextual equivalence can be characterised by some form of bisimilarity, showing that there is a robust notion of program equivalence. Program equivalence is indeed essential for studying program transformations and optimisations at work in compilers.

Finally, there is a sort of meta-principle: the more principles are connected, the better. For instance, it is desirable that evaluation in the calculus corresponds to cut-elimination in some logical interpretation of the calculus. Denotations are usually at least required to be *adequate* with respect to the rewriting: the denotation of a term is non-degenerated if and only if its evaluation terminates. Additionally, denotations are *fully abstract* if they reflect contextual equivalence. And implementations have to work within an overhead that respects the intended cost semantics. Ideally, all principles are satisfied and perfectly interconnected.

Of course, some specific cases may drop some requirements—for instance, a probabilistic λ-calculus would not be confluent—some properties may also be strengthened—for instance, equality may be characterised via a separation theorem akin to Bohm's—and other principles may be added—categorical semantics, graphical representations, etc.

What is usually considered *the* λ-calculus, is, in our terminology, the strong CbN λ-calculus with open terms, and all points of the benchmark have been studied for it. Plotkin's original formulation of CbV [45], conceived for weak evaluation and closed terms and here referred to as *Closed CbV*, also boldly passes the benchmark. Unfortunately Plotkin's setting fails the benchmark as soon as it is extended to open terms, which is required when using CbV for implementing

[4] Here *reasonable* is a technical word meaning that the cost model is polynomially equivalent to the one of Turing machines.

proof assistants, see Grégoire and Leroy's [29]. Typically, denotations are no longer adequate, as first noticed by Paolini and Ronchi della Rocca [48], and there is a mismatch between evaluation in the calculus and cut-elimination in its linear logic interpretation, as shown by Accattoli [1]. The failure can be observed also at other levels not covered by our benchmark, *e.g.* the incompleteness of CPS translations, already noticed by Plotkin himself [45].

Benchmarking open call-by-value. The problematic interaction of CbV and open terms is well known, and the fault is usually given to the rewriting—the operational semantics has to be changed somehow. The literature contains a number of proposals for extensions of CbV out of the closed world, some of which were introduced to solve the incompleteness of CPS translations. In [3], Accattoli and Guerrieri provided a comparative study of four extensions of Closed CbV (with weak evaluation on possibly open terms), showing that they have equivalent rewriting theories, are all adequate with respect to denotations, and share the same time cost models—these proposals have then to be considered as different incarnations of a more abstract framework, that they call *open call-by-value* (Open CbV). Accattoli, Sacerdoti Coen, and Guerrieri provided also a theory of implementations respecting the cost semantics [7,4], and a precise linear logic interpretation [1]. Thus, Open CbV passes the first five points of the benchmark.

This paper deepens the analysis of the fifth point, by refining the denotational understanding of Open CbV with a quantitative relationship with the rewriting and the cost model. We connect the size of type derivations for a term with its evaluation via rewriting, and the size of elements in its denotation with the size of its normal form, in a model coming from the linear logic interpretation of CbV and presented as a type system: Ehrhard's relational semantics for CbV [23].

The last point of the benchmark—contextual equivalence for Open CbV—was shown by Lassen to be a difficult question [39], and it is left to future work.

Multi types. Intersection types are one of the standard tools to study λ-calculi, mainly used to characterise termination properties—classical references are Coppo and Dezani [19,20], Pottinger [46], and Krivine [38]. In contrast to other type systems, they do not provide a logical interpretation, at least not as smoothly as for simple or polymorphic types—see Ronchi della Rocca and Roversi's [49] or Bono, Venneri, and Bettini's [9] for details. They are better understood, in fact, as syntactic presentations of denotational semantics: they are invariant under evaluation and type all and only terminating terms, thus naturally providing an adequate denotational semantics.

They are a flexible tool that can be formulated in various ways. A flavour that emerged in the last 10 years is that of *non-idempotent* intersection types, where the intersection $A \cap A$ is not equivalent to A. They were first considered by Gardner [26], and then Kfoury [37], Neergaard and Mairson [41], and de Carvalho [14,16] provided a first wave of works abut them—a survey can be found in Bucciarelli, Kesner, and Ventura's [12]. Non-idempotent intersections can be seen as multisets, which is why, to ease the language, we prefer to call them *multi types* rather than *non-idempotent intersection types*.

Multi types retain the denotational character of intersection types, and they actually refine it along two correlated lines. First, taking types with multiplicities gives rise to a *quantitative* approach, that reflects resource consumption in the evaluation of terms. Second, such a quantitative feature turns out to coincide exactly with the one at work in linear logic. Some care is needed here: multi types do not correspond to linear logic formulas, rather to the relational denotational semantics of linear logic (two seminal references for such a semantic are Girard's [28] and Bucciarelli and Ehrhard's [10]; see also [15,34])—similarly to intersection types, they provide a denotational rather than a logical interpretation.

An insightful use of multi types is de Carvalho's connection between the size of types and the size of normal forms, and between the size of type derivations and evaluation lengths for the CbN λ-calculus [16].

Types of fireballs. This paper develops a denotational analysis of Open CbV akin to de Carvalho's. There are two main translations of the λ-calculus into linear logic, due to Girard [27], the CbN one, that underlies de Carvalho's study, and the CbV one, that is explored here. The literature contains denotational semantics of CbV and also studies of multi types for CbV. The distinguishing feature of our study is the use of multi types to provide bounds on the number of evaluation steps and on the size of normal forms, which has never been done before for CbV, and that moreover we do it for the open case—the result for the closed case, refining Ehrhard's study, follows as a special case. Moreover, we provide a characterisation of types and type derivations that provide *exact* bounds, similarly to de Carvalho [14,16], Bernadet and Lengrand [8], and de Carvalho, Pagani, and Tortora de Falco [17], and along the lines of a very recent work by Accattoli, Graham-Lengrand, and Kesner [2], but using a slightly different approach.

Extracting exact bounds from the multi types system is however only half of the story. The other, subtler half is about tuning up the presentation of Open CbV as to accommodate as many points of the benchmark as possible. Our quantitative denotational inquire via multi types requires the following properties:

0. *Compositionality*: if two terms have the same type assignments, then the terms obtained by plugging them in the same context do so.
1. *Invariance under evaluation*: type assignments have to be stable by evaluation.
2. *Adequacy*: a term is typable if and only if it terminates.
3. *Elegant normal forms*: normal forms have a simple structure, so that the technical development is simple and intuitive.
4. *Number of steps*: type derivations have to provide the number of steps to evaluate to normal forms, and this number must be a reasonable cost model.
5. *Matching of sizes*: the size of normal forms has to be bounded by the size of their types.

While property 0 is not problematic (type systems/denotational models are conceived to satisfy it), it turns out that none of the incarnations of Open CbV studied by Accattoli and Guerrieri in [3] (namely, Paolini and Ronchi della Rocca's *fireball calculus* λ_{fire} [44,48,29,7], Accattoli and Paolini's *value substitution calculus*

λ_{vsub} [6,1], and Carraro and Guerrieri's *shuffling calculus* λ_{sh} [13,32,30,33,31])[5] satisfies all the properties 1–5 at the same time: λ_{fire} lacks property 1 (as shown here in Sect. 2); λ_{vsub} lacks property 3 (the inelegant characterisation of normal forms is in [6]); and λ_{sh}, which in [13] is shown to satisfy 1, 2, and partially 3, lacks properties 4 (the number of steps does not seem to be a reasonable cost model, see [3]) and 5 (see the end of Sect. 6 in this paper).

We then introduce the *split fireball calculus*, that is a minor variant of the fireball calculus λ_{fire}, isomorphic to it but integrating some features of the value substitution calculus λ_{vsub}, and satisfying all the requirements for our study. Thus, the denotational study follows smooth and natural, fully validating the design and the benchmark.

To sum up, our study adds new ingredients to the understanding of Open CbV, by providing a simple and quantitative denotational analysis via an adaptation of de Carvalho's approach, whose main features are:

1. *Split fireball calculus*: a new incarnation of Open CbV more apt to denotational studies, and conservative with respect to the other properties of the setting.
2. *Quantitative characterisation of termination*: proofs that typable terms are exactly the normalising ones, and that types and type derivations provide bounds on the size of normal forms and on evaluation lengths.
3. *Tight derivations and exact bounds*: a class of type derivations that provide the exact length of evaluations, and such that the types in the final judgements provide the exact size of normal forms.

Related work. Classical studies of the denotational semantics of Closed CbV are due to Sieber [50], Fiore and Plotkin [25], Honda and Yoshida [35], and Pravato, Ronchi della Rocca and Roversi [47]. A number of works rely on multi types or relational semantics to study property of programs and proofs. Among them, Ehrhard's [23], Diaz-Caro, Manzonetto, and Pagani's [22], Carraro and Guerrieri's [13], Ehrhard and Guerrieri's [24], and Guerrieri's [31] deal with CbV, while de Carvalho's [14,16], Bernadet and Lengrand's [8], de Carvalho, Pagani, and Tortora de Falco's [17], Accattoli, Graham-Lengrand, and Kesner's [2] provide exact bounds. Further related work about multi types is by Bucciarelli, Ehrhard, and Manzonetto [11], de Carvalho and Tortora de Falco [18], Kesner and Vial [36], and Mazza, Pellissier, and Vial [40]—this list is not exhaustive.

(No) Proofs. All proofs are in the Appendix of [5], the long version of this paper.

2 The Rise of Fireballs

In this section we recall the fireball calculus λ_{fire}, the simplest presentation of Open CbV. For the issues of Plotkin's setting with respect to open terms and for alternative presentations of Open CbV, see Accattoli and Guerrieri's [3].

[5] In [3] a fourth incarnation, the *value sequent calculus* (a fragment of Curien and Herbelin's $\bar{\lambda}\tilde{\mu}$ [21]), is proved isomorphic to a fragment of λ_{vsub}, which then subsumes it.

$$
\begin{array}{rl}
\textsc{Terms} & t,u,s,r ::= x \mid \lambda x.t \mid tu \\
\textsc{Values} & v,v',v'' ::= x \mid \lambda x.t \\
\textsc{Fireballs} & f,f',f'' ::= v \mid i \\
\textsc{Inert terms} & i,i',i'' ::= x f_1 \dots f_n \quad n>0 \\
\textsc{Right evaluation contexts} & C ::= \langle\cdot\rangle \mid tC \mid Cf
\end{array}
$$

$$
\begin{array}{cc}
\textsc{Rule at top level} & \textsc{Contextual closure} \\
(\lambda x.t)v \;\mapsto_{\beta_v}\; t\{x\leftarrow v\} & C\langle t\rangle \to_{\beta_v} C\langle u\rangle \;\text{ if } t \mapsto_{\beta_v} u \\
(\lambda x.t)i \;\mapsto_{\beta_i}\; t\{x\leftarrow i\} & C\langle t\rangle \to_{\beta_i} C\langle u\rangle \;\text{ if } t \mapsto_{\beta_i} u
\end{array}
$$

$$
\textsc{Reduction} \qquad \to_{\beta_f} \;:=\; \to_{\beta_v} \cup \to_{\beta_i}
$$

Fig. 1. The fireball calculus λ_{fire}.

The fireball calculus was introduced without a name and studied first by Paolini and Ronchi Della Rocca in [44,48]. It has then been rediscovered by Grégoire and Leroy in [29] to improve the implementation of Coq, and later by Accattoli and Sacerdoti Coen in [7] to study cost models, where it was also named. We present it following [7], changing only inessential, cosmetic details.

The fireball calculus. The fireball calculus λ_{fire} is defined in Fig. 1. The idea is that the values of the CbV λ-calculus — *i.e.* abstractions and variables — are generalised to *fireballs*, by extending variables to more general *inert terms*. Actually fireballs (noted $f, f', \dots$) and inert terms (noted $i, i', \dots$) are defined by mutual induction (in Fig. 1). For instance, x and $\lambda x.y$ are fireballs as values, while $y(\lambda x.x)$, xy, and $(z(\lambda x.x))(zz)(\lambda y.(zy))$ are fireballs as inert terms.

The main feature of inert terms is that they are open, normal, and that when plugged in a context they cannot create a redex, hence the name. Essentially, they are the *neutral terms* of Open CbV. In Grégoire and Leroy's presentation [29], inert terms are called *accumulators* and fireballs are simply called values.

Terms are always identified up to α-equivalence and the set of free variables of a term t is denoted by $\mathsf{fv}(t)$. We use $t\{x\leftarrow u\}$ for the term obtained by the capture-avoiding substitution of u for each free occurrence of x in t.

Variables are, morally, both values and inert terms. In [7] they were considered as inert terms, while here, for minor technical reasons we prefer to consider them as values and not as inert terms—the change is inessential.

Evaluation rules. Evaluation is given by *call-by-fireball* β-reduction $\to_{\beta_f}$: the β-rule can fire, *lighting up* the argument, only if the argument is a fireball (*fireball* is a catchier version of *fire-able term*). We actually distinguish two sub-rules: one that *lights up* values, noted $\to_{\beta_v}$, and one that *lights up* inert terms, noted $\to_{\beta_i}$ (see Fig. 1). Note that evaluation is *weak*: it does not reduce under abstractions.

We endow the calculus with the (deterministic) right-to-left evaluation strategy, defined via right evaluation contexts C—note the production Cf, forcing the right-to-left order. A more general calculus is defined in [3], for which the right-to-left strategy is shown to be complete. The left-to-right strategy, often adopted in the literature on Closed CbV, is also complete, but in the open case the right-to-left one has stronger invariants that lead to simpler abstract machines

(see [4]), which is why we adopt it here. We omit details about the rewriting theory of the fireball calculus because our focus here is on denotational semantics.

Properties. A famous key property of Closed CbV (whose evaluation is exactly $\to_{\beta_v}$) is *harmony*: given a closed term t, either it diverges or it evaluates to an abstraction, *i.e.* t is β_v-normal if and only if t is an abstraction. The fireball calculus λ_{fire} satisfies an analogous property in the (more general) *open* setting by replacing abstractions with fireballs (Prop. 1.1). Moreover, the fireball calculus is a *conservative extension* of Closed CbV: on closed terms it collapses on Closed CbV (Prop. 1.2). No other presentation of Open CbV has these good properties.

Proposition 1 (Distinctive properties of λ_{fire}). *Let t be a term.*

1. Open harmony*: t is β_f-normal if and only if t is a fireball.*
2. Conservative open extension*: $t \to_{\beta_f} u$ if and only if $t \to_{\beta_v} u$, when t is closed.*

Example 2. Let $t \coloneqq (\lambda z.z(yz))\lambda x.x$. Then, $t \to_{\beta_f} (\lambda x.x)(y\,\lambda x.x) \to_{\beta_f} y\,\lambda x.x$, where the final term $y\,\lambda x.x$ is a fireball (and β_f-normal).

The key property of inert terms is summarised by the following proposition: substitution of inert terms does not create or erase β_f-redexes, and hence can always be avoided. It plays a role in Sect. 4.

Proposition 3 (Inert substitutions and evaluation commute). *Let t, u be terms, i be an inert term. Then, $t \to_{\beta_f} u$ if and only if $t\{x\leftarrow i\} \to_{\beta_f} u\{x\leftarrow i\}$.*

With general terms (or even fireballs) instead of inert ones, evaluation and substitution do not commute, in the sense that both directions of Prop. 3 do not hold. Direction $\Leftarrow$ is false because substitution can create β_f-redexes, as in $(xy)\{x\leftarrow\lambda z.z\} = (\lambda z.z)y$; direction $\Rightarrow$ is false because substitution can erase β_f-redexes, as in $((\lambda x.z)(xx))\{x\leftarrow\delta\} = (\lambda x.z)(\delta\delta)$ where $\delta \coloneqq \lambda y.yy$.[6]

3 The Fall of Fireballs

Here we introduce Ehrhard's multi type system for CbV [23] and show that—with respect to it—the fireball calculus λ_{fire} fails the denotational test of the benchmark sketched in Sect. 1. This is an issue of λ_{fire}: to our knowledge, all denotational models that are adequate for (some variant of) CbV are not invariant under the evaluation rules of λ_{fire}, because of the rule $\to_{\beta_i}$ substituting inert terms[7].

In the next sections we shall use this type system, while the failure is not required for the main results of the paper, and may be skipped on a first reading.

[6] As well-known, Prop. 3 with ordinary (*i.e.* CbN) β-reduction $\to_\beta$ instead of $\to_{\beta_f}$ and general terms instead of inert ones holds only in direction $\Rightarrow$.

[7] Clearly, any denotational model for the CbN λ-calculus is invariant under β_f-reduction (since $\to_{\beta_f} \subseteq \to_\beta$), but there is no hope that it could be adequate for the fireball calculus. Indeed, such a model would identify the interpretations of $(\lambda x.y)\Omega$ (where Ω is a diverging term and $x \neq y$) and y, but in a CbV setting these two terms have a completely different behaviour: y is normal, whereas $(\lambda x.y)\Omega$ cannot normalise.

Relational semantics. We analyse the failure considering a concrete and well-known denotational model for CbV: *relational semantics*. For Plotkin's original CbV λ-calculus, it has been introduced by Ehrhard [23]. More generally, relational semantics provides a sort of canonical model of linear logic [27,10,34,15], and Ehrhard's model is the one obtained by representing the CbV λ-calculus into linear logic, and then interpreting it according to the relational semantics. It is also strongly related to other denotational models for CbV based on linear logic such as Scott domains and coherent semantics [23,47], and it has a well-studied CbN counterpart [14,16,11,43,42,40,2].

Relational semantics for CbV admits a nice syntactic presentation as a *multi type system* (aka non-idempotent intersection types), introduced right next. This type system, first studied by Ehrhard in [23], is nothing but the CbV version of de Carvalho's System R for CbN λ-calculus [14,16].

Multi types. Multi types and linear types are defined by mutual induction:

$$\begin{array}{ll} \textsc{Linear types} & L, L' ::= M \multimap N \\ \textsc{Multi types} & M, N ::= [L_1, \dots, L_n] \quad (\text{with } n \in \mathbb{N}) \end{array}$$

where $[L_1, \dots, L_n]$ is our notation for multisets. Note the absence of base types: their role is played by the *empty multiset* $[\,]$ (obtained for $n = 0$), that we rather note $\mathbf{0}$ and refer to as *the empty (multi) type*. A multi type $[L_1, \dots, L_n]$ has to be intended as a conjunction $L_1 \wedge \cdots \wedge L_n$ of linear types $L_1, \dots, L_n$, for a commutative and associative conjunction connective $\wedge$ that is not idempotent (morally a tensor $\otimes$) and whose neutral element is $\mathbf{0}$.

The intuition is that a linear type corresponds to a single use of a term t, and that t is typed with a multiset M of n linear types if it is going to be used (at most) n times. The meaning of *using a term* is not easy to define precisely. Roughly, it means that if t is part of a larger term u, then (at most) n copies of t shall end up in evaluation position during the evaluation of u. More precisely, the n copies shall end up in evaluation positions where they are applied to some terms.

The derivation rules for the multi types system are in Fig. 2—they are exactly the same as in [23]. In this system, *judgements* have the shape $\Gamma \vdash t : M$ where t is a term, M is a multi type and Γ is a *type context*, that is, a total function from variables to multi types such that the set $\mathsf{dom}(\Gamma) := \{x \mid \Gamma(x) \neq \mathbf{0}\}$ is finite. Note that terms are always assigned a multi type, and never a linear type—this is dual to what happens in de Carvalho's System R for CbN [14,16].

The application rule has a multiplicative formulation (in linear logic terminology), as it collects the type contexts of the two premises. The involved operation is the *sum of type contexts* $\Gamma \uplus \Delta$, that is defined as $(\Gamma \uplus \Delta)(x) := \Gamma(x) \uplus \Delta(x)$, where the $\uplus$ in the RHS stands for the multiset sum. A type context Γ such that $\mathsf{dom}(\Gamma) \subseteq \{x_1, \dots, x_n\}$ with $x_i \neq x_j$ and $\Gamma(x_i) = M_i$ for all $1 \leq i \neq j \leq n$ is often written as $\Gamma = x_1 : M_1, \dots, x_n : M_n$. Note that the sum of type contexts $\uplus$ is commutative and associative, and its neutral element is the type context Γ such that $\mathsf{dom}(\Gamma) = \emptyset$, which is called the *empty type context* (all types in Γ are $\mathbf{0}$). The notation $\pi \rhd \Gamma \vdash t : M$ means that π is a *type derivation* π (*i.e.* a tree constructed using the rules in Fig. 2) with conclusion the judgement $\Gamma \vdash t : M$.

$$\frac{}{x:M \vdash x:M}\ \mathsf{ax} \qquad \frac{\Gamma \vdash t:[M \multimap N] \qquad \Delta \vdash u:M}{\Gamma \uplus \Delta \vdash tu:N}\ @$$

$$\frac{\Gamma_1, x:M_1 \vdash t:N_1 \qquad \overset{n\in\mathbb{N}}{\vdots} \qquad \Gamma_n, x:M_n \vdash t:N_n}{\Gamma_1 \uplus \cdots \uplus \Gamma_n \vdash \lambda x.t:[M_1 \multimap N_1, \ldots, M_n \multimap N_n]}\ \lambda$$

Fig. 2. Multi types system for Plotkin's CbV λ-calculus [23].

Intuitions: the empty type $\mathbf{0}$. Before digging into technical details let us provide some intuitions. A key type specific to the CbV setting is the empty multiset $\mathbf{0}$, also known as the empty (multi) type. The idea is that $\mathbf{0}$ is the type of terms that can be erased. To understand its role in CbV, we first recall its role in CbN.

In the CbN multi type system [14,16,2] every term, even a diverging one, is typable with $\mathbf{0}$. On the one hand, this is correct, because in CbN every term can be erased, and erased terms can also be divergent, because they are never evaluated. On the other hand, adequacy is formulated with respect to non-empty types: a term terminates if and only if it is typable with a non-empty type.

In CbV, instead, terms have to be evaluated before being erased. And, of course, their evaluation has to terminate. Therefore, terminating terms and erasable terms coincide. Since the multi type system is meant to characterise terminating terms, in CbV a term is typable if and only if it is typable with $\mathbf{0}$, as we shall prove in Sect. 8. Then the empty type is not a degenerate type, as in CbN, it rather is *the* type, characterising (adequate) typability altogether.

Note that, in particular, in a typing judgement $\Gamma \vdash e:M$ the type context Γ may give the empty type to a variable x occurring in e, as for instance in the axiom $x:\mathbf{0} \vdash x:\mathbf{0}$—this may seem very strange to people familiar with CbN multi types. We hope that instead, according to the provided intuition that $\mathbf{0}$ is the type of termination, it would rather seem natural.

The model. The idea to build the denotational model from the type system is that the interpretation (or semantics) of a term is simply the set of its type assignments, *i.e.* the set of its derivable types together with their type contexts. More precisely, let t be a term and $x_1, \ldots, x_n$ (with $n \geq 0$) be pairwise distinct variables. If $\mathsf{fv}(t) \subseteq \{x_1, \ldots, x_n\}$, we say that the list $\vec{x} = (x_1, \ldots, x_n)$ is *suitable for* t. If $\vec{x} = (x_1, \ldots, x_n)$ is suitable for t, the (*relational*) *semantics of* t *for* $\vec{x}$ is

$$[\![t]\!]_{\vec{x}} := \{((M_1, \ldots, M_n), N) \mid \exists\, \pi \rhd x_1:M_1, \ldots, x_n:M_n \vdash t:N\}\,.$$

Ehrhard proved that this is a denotational model for Plotkin's CbV λ-calculus [23, p. 267], in the sense that the semantics of a term is invariant under β_v-reduction.

Theorem 4 (Invariance for $\to_{\beta_v}$, [23]). *Let t and u be two terms and $\vec{x} = (x_1, \ldots, x_n)$ be a suitable list of variables for t and u. If $t \to_{\beta_v} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.*

Note that terms are not assumed to be closed. Unfortunately, relational semantics is not a denotational model of the fireball calculus λ_{fire}: Thm. 4 does not hold if we replace $\to_{\beta_v}$ with $\to_{\beta_i}$ (and hence with $\to_{\beta_f}$), as we show in the following example—the reader can skip it on a first reading.

Example 5 (On a second reading: non-invariance of multi types in the fireball calculus). Consider the fireball step $(\lambda z.y)(xx) \to_{\beta_f} y$, where the inert sub-term xx is erased. Let us construct the interpretations of the terms $(\lambda z.y)(xx)$ and y. All type derivations for xx are as follows (M and N are arbitrary multi types):

$$\pi_{M,N} = \dfrac{\dfrac{}{x : [M \multimap N] \vdash x : [M \multimap N]}\,\mathsf{ax} \qquad \dfrac{}{x : M \vdash x : M}\,\mathsf{ax}}{x : [M \multimap N] \uplus M \vdash xx : N}\,@$$

Hence, all type derivations for $(\lambda z.y)(xx)$ and y have the following forms:

$$\dfrac{\dfrac{\dfrac{}{y : N \vdash y : N}\,\mathsf{ax}}{y : N \vdash \lambda z.y : [\mathbf{0} \multimap N]}\,\lambda \qquad \begin{array}{c}\vdots\\ \pi_{M,\mathsf{0}}\\ \vdots\end{array}}{x : [M \multimap \mathbf{0}] \uplus M, y : N \vdash (\lambda z.y)(xx) : N}\,@ \qquad \dfrac{}{x : \mathbf{0}, y : N \vdash y : N}\,\mathsf{ax}$$

Therefore,

$$\begin{aligned} [\![(\lambda z.y)(xx)]\!]_{x,y} &= \{(([M \multimap \mathbf{0}] \uplus M, N), N) \mid M, N \text{ multi types}\} \\ [\![y]\!]_{x,y} &= \{((\mathbf{0}, N), N) \mid N \text{ multi type}\} \end{aligned}$$

To sum up, in the fireball calculus $(\lambda z.y)(xx) \to_{\beta_f} y$, but $[\![(\lambda z.y)(xx)]\!]_{x,y} \not\subseteq [\![y]\!]_{x,y}$ as $(([\mathbf{0} \multimap \mathbf{0}], \mathbf{0}), \mathbf{0}) \in [\![(\lambda z.y)(xx)]\!]_{x,y} \setminus [\![y]\!]_{x,y}$, and $[\![y]\!]_{x,y} \not\subseteq [\![(\lambda z.y)(xx)]\!]_{x,y}$ because $((\mathbf{0}, \mathbf{0}), \mathbf{0}) \in [\![y]\!]_{x,y} \setminus [\![(\lambda z.y)(xx)]\!]_{x,y}$.

An analogous problem affects the reduction step $(\lambda z.zz)(xx) \to_{\beta_f} (xx)(xx)$, where the inert term xx is instead duplicated. In general, all counterexamples to the invariance of the relational semantics under β_f-reduction are due to β_i-reduction, when the argument of the fired β_f-redex is an inert term that is erased or duplicated. Intuitively, to fix this issue, we should modify the syntax and operational semantics of λ_{fire} in such a way that the β_i-step destroys the β-redex without erasing nor duplicating its inert argument: Prop. 3 guarantees that this modification is harmless. This new presentation of λ_{fire} is in the next section.

Remark 6 (On a second reading: additional remarks about relational semantics).

1. Relational semantics is invariant for Plotkin's CbV even in presence of *open* terms, but *it no longer is an adequate model*: the term $(\lambda y.\delta)(xx)\delta$ (where $\delta \coloneqq \lambda z.zz$) has an empty semantics (*i.e.* is not typable in the multi type system of Fig. 2) but it is β_v-normal. Note that, instead, it diverges in λ_{fire} because a β_i-step "unblocks" it: $(\lambda y.\delta)(xx)\delta \to_{\beta_i} \delta\delta \to_{\beta_v} \delta\delta \to_{\beta_v} \dots$
2. Even though it is not a denotational model for the fireball calculus, relational semantics is *adequate* for it, in the sense that a term is typable in the multi types system of Fig. 2 if and only if it β_f-normalises. This follows from two results involving the shuffling calculus, an extension of Plotkin's CbV that is another presentation of Open CbV:
   - the adequacy of the relational semantics for the shuffling calculus [13,31];

Terms, Values, Fireballs, Inert terms, Right ev. contexts	as for the fireball calculus λ_{fire}
Environments	$E ::= \epsilon \mid [x \leftarrow i] : E$
Programs	$p ::= (t, E)$
Rules	$(C\langle(\lambda x.t)\ v\rangle, E) \to_{\beta_v} (C\langle t\{x \leftarrow v\}\rangle, E)$ $(C\langle(\lambda x.t)\ i\rangle, E) \to_{\beta_i} (C\langle t\rangle, [x \leftarrow i] : E)$
Reduction	$\to_{\beta_f} := \to_{\beta_v} \cup \to_{\beta_i}$

Fig. 3. The split fireball calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$.

– the equivalence of the fireball calculus λ_{fire} and shuffling calculus λ_{sh} from the termination point of view, *i.e.* a term normalises in one calculus if and only if it normalises in the other one [3].

Unfortunately, the shuffling calculus λ_{sh} has issues with respect to the quantitative aspects of the semantics (it is unknown whether its number of steps is a reasonable cost model [3]; the size of λ_{sh}-normal forms is not bounded by the size of their types, as we show in Ex. 16), which instead better fit the fireball calculus λ_{fire}. This is why in the next section we slightly modify λ_{fire}, rather than switching to λ_{sh}.

4 Fireballs Reloaded: the Split Fireball Calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$

This section presents the *split fireball calculus* $\mathsf{Split}\lambda_{\mathsf{fire}}$, that is the refinement of the fireball calculus λ_{fire} correcting the issue explained in the previous section (Ex. 5), namely the non-invariance of type assignments by evaluation.

The calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$ is defined in Fig. 3. The underlying idea is simple: the problem with the fireball calculus is the substitution of inert terms, as discussed in Ex. 5; but some form of β_i-step is needed to get the adequacy of relational semantics in presence of open terms, as shown in Rmk. 6. Inspired by Prop. 3, the solution is to keep trace of the inert terms involved in β_i-steps in an auxiliary environment, without substituting them in the body of the abstraction. Therefore, we introduce the syntactic category of *programs* p, that are terms with an *environment* E, which in turn is a list of explicit (*i.e.* delayed) substitutions paring variables and inert terms. We use *expressions* $e, e', \ldots$ to refer to the union of terms and programs. Note the new form of the rewriting rule $\to_{\beta_i}$, that does not substitute the inert term and rather adds an entry to the environment. Apart from storing inert terms, the environment does not play any active role in β_f-reduction for $\mathsf{Split}\lambda_{\mathsf{fire}}$. Even though $\to_{\beta_f}$ is a binary relation on programs, we use '*normal expression*' to refer to either a normal (with respect to $\to_{\beta_f}$) program or a term t such that the program (t, E) is normal (for any environment E).

The good properties of the fireball calculus are retained. Harmony in $\mathsf{Split}\lambda_{\mathsf{fire}}$ takes the following form (for arbitrary fireball f and environment E):

Proposition 7 (Harmony). *A program p is normal if and only if $p = (f, E)$.*

So, an expression is normal iff it is a fireball f or a program of the form (f, E).

Conservativity with respect to the closed case is also immediate, because in the closed case the rule $\to_{\beta_i}$ never fires and so the environment is always empty.

On a second reading: no open size explosion. Let us mention that avoiding the substitution of inert terms is also natural at the implementation / cost model level, as substituting them causes *open size explosion*, an issue studied at length in previous work on the fireball calculus [7,4]. Avoiding the substitution of inert terms altogether is in fact what is done by the other incarnations of Open CbV, as well as by abstract machines. The split fireball calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$ can in fact be seen as adding the environment of abstract machines but without having to deal with the intricacies of decomposed evaluation rules. It can also be seen as the (open fragment of) Accattoli and Paolini's *value substitution calculus* [6], where indeed inert terms are never substituted. In particular, it is possible to prove that the normal forms of the split fireball calculus are isomorphic to those of the value substitution up to its structural equivalence (see [3] for the definitions).

On a second reading: relationship with the fireball calculus. The split and the (plain) fireball calculus are isomorphic at the rewriting level. To state the relationship we need the concept of *program unfolding* $(t, E)\!\downarrow$, that is, the term obtained by substituting the inert terms in the environment E into the main term t:

$$(t, \epsilon)\!\downarrow \coloneqq t \qquad\qquad (t, [y{\leftarrow}i] : E)\!\downarrow \coloneqq (t\{x{\leftarrow}i\}, E)\!\downarrow$$

From the commutation of evaluation and substitution of inert terms in the fireball calculus (Prop. 3), it follows that normal programs (in $\mathsf{Split}\lambda_{\mathsf{fire}}$) unfold to normal terms (in λ_{fire}), that is, fireballs. Conversely, every fireball can be seen as a normal program with respect to the empty environment.

For evaluation, the same commutation property easily gives the following strong bisimulation between the split $\mathsf{Split}\lambda_{\mathsf{fire}}$ and the plain λ_{fire} fireball calculi.

Proposition 8 (Strong bisimulation). *Let p be a program (in $\mathsf{Split}\lambda_{\mathsf{fire}}$).*

1. Split to plain*: if $p \to_{\beta_f} q$ then $p\!\downarrow \to_{\beta_f} q\!\downarrow$.*
2. Plain to split*: if $p\!\downarrow \to_{\beta_f} u$ then there exists q such that $p \to_{\beta_f} q$ and $q\!\downarrow = u$.*

It is then immediate that termination in the two calculi coincide, as well as the number of steps to reach a normal form. Said differently, the split fireball calculus can be seen as an *isomorphic* refinement of the fireball calculus.

5 Multi Types for $\mathsf{Split}\lambda_{\mathsf{fire}}$

The multi type system for the split fireball calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$ is the natural extension to terms with environments of the multi type system for Plotkin's CbV λ-calculus seen in Sect. 2. Multi and linear types are the same. The only novelty is that now judgements type expressions, not only terms, hence we add two new rules for the two cases of environment, es_ϵ and $\mathsf{es}_@$, see Fig. 4. Rule es_ϵ is trivial, it is simply the coercion of a term to a program with an empty environment. Rule $\mathsf{es}_@$ uses the *append* operation $E@[x{\leftarrow}i]$ that appends an entry $[x{\leftarrow}i]$ to the end of an environment E, formally defined as follows:

$$\epsilon@[x{\leftarrow}i] \coloneqq [x{\leftarrow}i] \qquad\qquad ([y{\leftarrow}i'] : E)@[x{\leftarrow}i] \coloneqq [y{\leftarrow}i'] : (E@[x{\leftarrow}i])$$

We keep all the notations already used for multi types in Sect. 3.

$$\frac{}{x:M \vdash x:M}\ \mathsf{ax} \qquad \frac{\Gamma \vdash t:[M \multimap N] \qquad \Delta \vdash u:M}{\Gamma \uplus \Delta \vdash tu:N}\ @$$

$$\frac{\Gamma_1, x:M_1 \vdash t:N_1 \qquad \overset{n\in\mathbb{N}}{\cdots} \qquad \Gamma_n, x:M_n \vdash t:N_n}{\Gamma_1 \uplus \cdots \uplus \Gamma_n \vdash \lambda x.t:[M_1 \multimap N_1, \ldots, M_n \multimap N_n]}\ \lambda$$

$$\frac{\Gamma \vdash t:M}{\Gamma \vdash (t,\epsilon):M}\ \mathsf{es}_\epsilon \qquad \frac{\Gamma, x:M \vdash (t,E):N \qquad \Delta \vdash i:M}{\Gamma \uplus \Delta \vdash (t, E@[x\leftarrow i]):N}\ \mathsf{es}_@$$

Fig. 4. Multi types system for the split fireball calculus.

Sizes, and basic properties of typing. For our quantitative analyses, we need the notions of size for terms, programs and type derivations.

The *size* $|t|$ *of a term* t is the number of its applications not under the scope of an abstraction. The *size* $|(t,E)|$ *of a program* (t,E) is the size of t plus the size of the (inert) terms in the environment E. Formally, they are defined as follows:

$$|v| \coloneqq 0 \quad |tu| \coloneqq |t| + |u| + 1 \qquad |(t,\epsilon)| \coloneqq |t| \quad |(t, E@[x\leftarrow i])| \coloneqq |(t,E)| + |i|$$

The *size* $|\pi|$ *of a type derivation* π is the number of its @ rules.

The proofs of the next basic properties of type derivations are straightforward.

Lemma 9 (Free variables in typing). *If* $\pi \rhd \Gamma \vdash e:M$ *then* $\mathsf{dom}(\Gamma) \subseteq \mathsf{fv}(e)$.

The next lemma collects some basic properties of type derivations for values.

Lemma 10 (Typing of values). *Let* $\pi \rhd \Gamma \vdash v:M$ *be a type derivation for a value* v*. Then,*

1. Empty multiset implies nul size*: if* $M = \mathbf{0}$ *then* $\mathsf{dom}(\Gamma) = \emptyset$ *and* $|\pi| = 0 = |v|$.
2. Multiset splitting*: if* $M = N \uplus O$*, then there are two type contexts* Δ *and* Π *and two type derivations* $\sigma \rhd \Delta \vdash v:N$ *and* $\rho \rhd \Pi \vdash v:O$ *such that* $\Gamma = \Delta \uplus \Pi$ *and* $|\pi| = |\sigma| + |\rho|$.
3. Empty judgement*: there is a type derivation* $\sigma \rhd \vdash v:\mathbf{0}$.
4. Multiset merging*: for any two type derivations* $\pi \rhd \Gamma \vdash v:M$ *and* $\sigma \rhd \Delta \vdash v:N$ *there is a type derivation* $\rho \rhd \Gamma \uplus \Delta \vdash v:M \uplus N$ *such that* $|\rho| = |\pi| + |\sigma|$.

The next two sections prove that the multi type system is correct (Sect. 6) and complete (Sect. 7) for termination in the split fireball calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$, also providing bounds for the length $|d|$ of a normalising evaluation d and for the size of normal forms. At the end of Sect. 7 we discuss the adequacy of the relational model induced by this multi type system, with respect to $\mathsf{Split}\lambda_{\mathsf{fire}}$. Sect. 8 characterises types and type derivations that provide exact bounds.

6 Correctness

Here we prove correctness (Thm. 14) of multi types for $\mathsf{Split}\lambda_{\mathsf{fire}}$, refined with quantitative information: if a term is typable then it terminates, and the type derivation provides bounds for both the number of steps to normal form and the size of the normal form. After the correctness theorem we show that even types by themselves—without the derivation—bound the size of normal forms.

Correctness. The proof technique is standard. Correctness is obtained from subject reduction (Prop. 13) plus a property of typings of normal forms (Prop. 11).

Proposition 11 (Type derivations bound the size of normal forms). *Let $\pi \rhd \Gamma \vdash e : M$ be a type derivation for a normal expression e. Then $|e| \leq |\pi|$.*

As it is standard in the study of type systems, subject reduction requires a substitution lemma for typed terms, here refined with quantitative information.

Lemma 12 (Substitution). *Let $\pi \rhd \Gamma, x : N \vdash t : M$ and $\sigma \rhd \Delta \vdash v : N$ (where v is a value). Then there exists $\rho \rhd \Gamma \uplus \Delta \vdash t\{x \leftarrow v\} : M$ such that $|\rho| = |\pi| + |\sigma|$.*

The key point of the next *quantitative* subject reduction property is the fact that the size of the derivation decreases by *exactly* 1 at each evaluation step.

Proposition 13 (Quantitative subject reduction). *Let p and p' be programs and $\pi \rhd \Gamma \vdash p : M$ be a type derivation for p. If $p \to_{\beta_f} p'$ then $|\pi| > 0$ and there exists a type derivation $\pi' \rhd \Gamma \vdash p' : M$ such that $|\pi'| = |\pi| - 1$.*

Correctness now follows as an easy induction on the size of the type derivation, which bounds both the length $|d|$ of the—normalising—evaluation d (*i.e.* the number of β_f-steps in d) by Prop. 13, and the size of the normal form by Prop. 11.

Theorem 14 (Correctness). *Let $\pi \rhd \Gamma \vdash p : M$ be a type derivation. Then there exist a normal program q and an evaluation $d \colon p \to_{\beta_f}^* q$ with $|d| + |q| \leq |\pi|$.*

Types bound the size of normal forms. In our multi type system, not only type derivations but also multi types provide quantitative information, in this case on the size of normal forms.

First, we need to define the size for multi types and type contexts, which is simply given by the number of occurrences of $\multimap$. Formally, the size of linear and multi types are defined by mutual induction by $|M \multimap N| := 1 + |M| + |N|$ and $|[L_1, \dots, L_n]| := \sum_{i=1}^{n} |L_i|$. Clearly, $|M| \geq 0$ and $|M| = 0$ if and only if $M = \mathbf{0}$.

Given a type context $\Gamma = x_1 : M_1, \dots, x_n : M_n$ we often consider the list of its types, noted $\underline{\Gamma} := (M_1, \dots, M_n)$. Since any list of multi types $(M_1, \dots, M_n)$ can be seen as extracted from a type context Γ, we use the notation $\underline{\Gamma}$ for lists of multi types. The size of a list of multi types is given by $|(M_1, \dots, M_n)| := \sum_{i=1}^{n} |M_i|$. Clearly, $\mathsf{dom}(\Gamma) = \emptyset$ if and only if $|\underline{\Gamma}| = 0$.

The quantitative information is that the size of types bounds the size of normal forms. In the case of inert terms a stronger bound actually holds.

Proposition 15 (Types bound the size of normal forms). *Let e be a normal expression. For any type derivation $\pi \rhd \Gamma \vdash e : M$, one has $|e| \leq |(\underline{\Gamma}, M)|$. If moreover e is an inert term, then $|e| + |M| \leq |\underline{\Gamma}|$.*

Example 16 (On a second reading: types, normal forms, and λ_{sh}). The fact that multi types bound the size of normal forms is a quite delicate result that holds in the split fireball calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$ but does not hold in other presentations of Open

CbV, like the shuffling calculus λ_{sh} [13,31], as we now show—this is one of the reasons motivating the introduction of $\mathsf{Split}\lambda_{\mathsf{fire}}$. Without going into the details of λ_{sh}, consider $t \coloneqq (\lambda z.z)(xx)$: it is normal for λ_{sh} but it — or, more precisely, the program $p \coloneqq (t, \epsilon)$ — is not normal for $\mathsf{Split}\lambda_{\mathsf{fire}}$, indeed $p \to^*_{\beta_f} (z, [y \leftarrow xx]) =: q$ and q is normal in $\mathsf{Split}\lambda_{\mathsf{fire}}$. Concerning sizes, $|t| = |p| = 2$ and $|q| = 1$. Consider the following type derivation for t (the type derivation $\pi_{\mathbf{0},\mathbf{0}}$ is defined in Ex. 5):

$$\dfrac{\dfrac{\dfrac{}{z:\mathbf{0} \vdash z:\mathbf{0}}\,\mathsf{ax}}{\vdash \lambda z.z:[\mathbf{0} \multimap \mathbf{0}]}\,\lambda \qquad \begin{array}{c}\vdots \pi_{\mathbf{0},\mathbf{0}} \\ x:[\mathbf{0} \multimap \mathbf{0}] \vdash xx:\mathbf{0}\end{array}}{x:[\mathbf{0} \multimap \mathbf{0}] \vdash (\lambda z.z)(xx):\mathbf{0}}\,@$$

So, $|t| = 2 > 1 = |([\mathbf{0} \multimap \mathbf{0}], \mathbf{0})|$, which gives a counterexample to Prop. 15 in λ_{sh}.

7 Completeness

Here we prove completeness (Thm. 20) of multi types for $\mathsf{Split}\lambda_{\mathsf{fire}}$, refined with quantitative information: if a term terminates then it is typable, and the quantitative information is the same as in the correctness theorem (Thm. 14 above). After that, we discuss the adequacy of the relational semantics induced by the multi type system, with respect to termination in $\mathsf{Split}\lambda_{\mathsf{fire}}$.

Completeness. The proof technique, again, is standard. Completeness is obtained by a subject expansion property plus the fact that all normal forms are typable.

Proposition 17 (Normal forms are typable).

1. Normal expression: *for any normal expression e, there exists a type derivation $\pi \rhd \Gamma \vdash e : M$ for some type context Γ and some multi type M.*
2. Inert term: *for any multi type N and any inert term i, there exists a type derivation $\sigma \rhd \Delta \vdash i : N$ for some type context Δ.*

In the proof of Prop. 17, the stronger statement for inert terms is required, to type a normal expression that is a program with non-empty environment.

For quantitative subject expansion (Prop. 19), which is dual to subject reduction (Prop. 13 above), we need an anti-substitution lemma that is the dual of the substitution one (Lemma 12 above).

Lemma 18 (Anti-substitution). *Let t be a term, v be a value, and $\pi \rhd \Gamma \vdash t\{x \leftarrow v\} : M$ be a type derivation. Then there exist two type derivations $\sigma \rhd \Delta, x : N \vdash t : M$ and $\rho \rhd \Pi \vdash v : N$ such that $\Gamma = \Delta \uplus \Pi$ and $|\pi| = |\sigma| + |\rho|$.*

Subject expansion follows. Dually to subject reduction, the size of the type derivation grows by *exactly* 1 along every expansion (*i.e.* along every anti-β_f-step).

Proposition 19 (Quantitative subject expansion). *Let p and p' be programs and $\pi' \rhd \Gamma \vdash p' : M$ be a type derivation for p'. If $p \to_{\beta_f} p'$ then there exists a type derivation $\pi \rhd \Gamma \vdash p : M$ for p such that $|\pi'| = |\pi| + 1$.*

Theorem 20 (Completeness). *Let $d : p \to^*_{\beta_f} q$ be a normalising evaluation. Then there is a type derivation $\pi \rhd \Gamma \vdash p : M$, and it satisfies $|d| + |q| \leq |\pi|$.*

Relational semantics. Subject reduction (Prop. 13) and expansion (Prop. 19) imply that the set of typing judgements of a term is invariant by evaluation, and so they provide a denotational model of the split fireball calculus (Cor. 21 below).

The definitions seen in Sect. 2 of the interpretation $[\![t]\!]_{\vec{x}}$ of a term with respect to a list $\vec{x}$ of suitable variables for t extends to the split fireball calculus by simply replacing terms with programs, with no surprises.

Corollary 21 (Invariance). *Let p and q be two programs and $\vec{x} = (x_1, \dots, x_n)$ be a suitable list of variables for p and q. If $p \to_{\beta_f} q$ then $[\![p]\!]_{\vec{x}} = [\![q]\!]_{\vec{x}}$.*

From correctness (Thm. 14) and completeness (Thm. 20) it follows that the relational semantics is adequate for the split fireball calculus $\mathsf{Split}\lambda_{\mathsf{fire}}$.

Corollary 22 (Adequacy). *Let p be a program and $\vec{x} = (x_1, \dots, x_n)$ be a suitable list of variables for p. The following are equivalent:*

1. Termination*: the evaluation of p terminates;*
2. Typability*: there is a type derivation $\pi \rhd \Gamma \vdash p : M$ for some Γ and M;*
3. Non-empty denotation*: $[\![p]\!]_{\vec{x}} \neq \emptyset$.*

Careful about the third point: it requires the interpretation to be non-empty—a program typable with the empty multiset $\mathbf{0}$ has a non-empty interpretation. Actually, a term is typable if and only if it is typable with $\mathbf{0}$, as we show next.

Remark 23. By Prop. 1.2 and 8, (weak) evaluations in Plotkin's original CbV λ-calculus λ_v, in the fireball calculus λ_{fire} and in its split variant $\mathsf{Split}\lambda_{\mathsf{fire}}$ coincide on closed terms. So, Cor. 22 says that relational semantics is adequate also for λ_v *restricted to closed terms* (but adequacy for λ_v fails on open terms, see Rmk. 6).

8 Tight Types and Exact Bounds

In this section we study a class of minimal type derivations, called *tight*, providing exact bounds for evaluation lengths and sizes of normal forms.

Typing values and inert terms. Values can always be typed with $\mathbf{0}$ in an empty type context (Lemma 10.3), by means of an axiom for variables or of a λ-rule with zero premises for abstractions. We are going to show that inert terms can also always be typed with $\mathbf{0}$. There are differences, however. First, the type context in general is not empty. Second, the derivations typing with $\mathbf{0}$ have a more complex structure, having sub-derivations for inert terms whose right-hand type might not be $\mathbf{0}$. It is then necessary, for inert terms, to consider a more general class of type derivations, that, as a special case, include derivations typing with $\mathbf{0}$.

First of all, we define two class of types:

Inert linear types	$L^i ::= \mathbf{0} \multimap N^i$
Inert multi types	$M^i, N^i ::= [L^i_1, \dots, L^i_n]$ (with $n \in \mathbb{N}$).

A type context Γ is *inert* if it assigns only inert multi types to variables.

In particular, the empty multi type $\mathbf{0}$ is inert (take $n = 0$), and hence the empty type context is inert. Note that inert multi types and inert multi contexts are closed under summation $\uplus$.

We also introduce two notions of type derivations, *inert* and *tight*. The tight ones are those we are actually interested in, but, as explained, for inert terms we need to consider a more general class of type derivations, the inert ones. Formally, given an expression e, a type derivation $\pi \rhd \Gamma \vdash e : M$ is

- *inert* if Γ is a inert type context and M is a inert multi type;
- *tight* if π is inert and $M = \mathbf{0}$;
- *nonempty* (resp. *empty*) if Γ is a non-empty (resp. empty) type context.

Note that tightness and inertness of type derivations depend only on the judgement in their conclusions. The general property is that inert terms admit a inert type derivation *for every inert multi type* M^i.

Lemma 24 (Inert typing of inert terms). *Let i be a inert term. For any inert multi type M^i there exists a nonempty inert type derivation $\pi \rhd \Gamma \vdash i : M^i$.*

Lemma 24 holds with respect to *all* inert multi types, in particular $\mathbf{0}$, so inert terms can be always typed with a nonempty *tight* derivation. Since values can be always typed with an empty tight derivation (Lemma 10.3), we can conclude:

Corollary 25 (Fireballs are tightly typable). *For any fireball f there exists a tight type derivation $\pi \rhd \Gamma \vdash f : \mathbf{0}$. Moreover, if f is a inert term then π is nonempty, otherwise f is a value and π is empty.*

By harmony (Prop. 7), it follows that any normal expression is tightly typable (Prop. 26 below). *Terminology*: a *coerced value* is a program of the form (v, ϵ).

Proposition 26 (Normal expressions are tightly typable). *Let e be a normal expression. Then there exists a tight derivation $\pi \rhd \Gamma \vdash e : \mathbf{0}$. Moreover, e is a value or a coerced value if and only if π is empty.*

Tight derivations and exact bounds. The next step is to show that tight derivations are minimal and provide exact bounds. Again, we have to detour through inert derivations for inert terms. And we need a further property of inert terms: if the type context is inert then the right-hand type is also inert.

Lemma 27 (Inert spreading on inert terms). *Let $\pi \rhd \Gamma \vdash i : M$ be a type derivation for a inert term i. If Γ is a inert type context then M and π are inert.*

Next, we prove that inert derivations provide exact bounds for inert terms.

Lemma 28 (Inert derivations are minimal and provide the exact size of inert terms). *Let $\pi \rhd \Gamma \vdash i : M^i$ be a inert type derivation for a inert term i. Then $|i| = |\pi|$ and $|\pi|$ is minimal among the type derivations of i.*

We can now extend the characterisation of sizes to all normal expressions, via tight derivations, refining Prop. 11.

Lemma 29 (Tight derivations are minimal and provide the exact size of normal forms). *Let $\pi \rhd \Gamma \vdash e : \mathbf{0}$ be a tight derivation and e be a normal expression. Then $|e| = |\pi|$ and $|\pi|$ is minimal among the type derivations of e.*

The bound on the size of normal forms using types rather than type derivations (Prop. 15) can also be refined: *tight* derivations end with judgements whose (inert) *type contexts* provide the *exact* size of normal forms.

Proposition 30 (Inert types and the exact size of normal forms). *Let e be a normal expression and $\pi \rhd \Gamma \vdash e : \mathbf{0}$ be a tight derivation. Then $|e| = |\underline{\Gamma}|$.*

Tightness and general programs. Via subject reduction and expansion, exact bounds can be extended to all normalisable programs. Tight derivations indeed induce refined correctness and completeness theorems replacing inequalities with equalities (see Thm. 31 and 32 below and compare them with Thm. 14 and 20 above, respectively): an *exact* quantitative information relates the length $|d|$ of evaluations, the size of normal forms and the size of *tight* type derivations.

Theorem 31 (Tight correctness). *Let $\pi \rhd \Gamma \vdash p : \mathbf{0}$ be a tight type derivation. Then there is a normalising evaluation $d \colon p \to^*_{\beta_f} q$ with $|\pi| = |d| + |q| = |d| + |\underline{\Gamma}|$. In particular, if $\mathsf{dom}(\Gamma) = \emptyset$, then $|\pi| = |d|$ and q is a coerced value.*

Theorem 32 (Tight completeness). *Let $d \colon p \to^*_{\beta_f} q$ be a normalising evaluation. Then there is a tight type derivation $\pi \rhd \Gamma \vdash p : \mathbf{0}$ with $|\pi| = |d| + |q| = |d| + |\underline{\Gamma}|$. In particular, if q is a coerced value, then $|\pi| = |d|$ and $\mathsf{dom}(\Gamma) = \emptyset$.*

Both theorems are proved analogously to their corresponding non-tight version (Thm. 14 and 20), the only difference is in the base case: here Lemma 29 provides an equality on sizes for normal forms, instead of the inequality given by Prop. 11 and used in the non-tight versions. The proof of tight completeness (Thm. 32) uses also that normal programs are *tightly* typable (Prop. 26).

9 Conclusions

This paper studies multi types for CbV weak evaluation. It recasts in CbV de Carvalho's work for CbN [14,16], building on a type system introduced by Ehrhard [23] for Plotkin's original CbV λ-calculus λ_v [45]. Multi types provide a denotational model that we show to be adequate for λ_v, but only when evaluating *closed* terms; and for Open CbV [3], an extension of λ_v where weak evaluation is on possibly *open* terms. More precisely, our main contributions are:

1. The formalism itself: we point out the issues with respect to subject reduction and expansion of the simplest presentation of Open CbV, the fireball calculus λ_{fire}, and introduce a refined calculus (isomorphic to λ_{fire}) that satisfies them.
2. The characterisation of termination both in a *qualitative* and *quantitative* way. Qualitatively, typable terms and normalisable terms coincide. Quantitatively, types provide bounds on the size of normal forms, and type derivations bound the number of evaluation steps to normal form.
3. The identification of a class of type derivations that provide *exact* bounds on evaluation lengths.

References

1. Accattoli, B.: Proof nets and the call-by-value λ-calculus. Theor. Comput. Sci. **606**, 2–24 (2015)
2. Accattoli, B., Graham-Lengrand, S., Kesner, D.: Tight typings and split bounds. In: ICFP 2018 (2018), to appear
3. Accattoli, B., Guerrieri, G.: Open Call-by-Value. In: APLAS 2016. pp. 206–226 (2016)
4. Accattoli, B., Guerrieri, G.: Implementing Open Call-by-Value. In: FSEN 2017. pp. 1–19 (2017)
5. Accattoli, B., Guerrieri, G.: Types of Fireballs (Extended Version). CoRR **abs/1808.10389** (2018)
6. Accattoli, B., Paolini, L.: Call-by-Value Solvability, revisited. In: FLOPS. pp. 4–16 (2012)
7. Accattoli, B., Sacerdoti Coen, C.: On the Relative Usefulness of Fireballs. In: LICS 2015. pp. 141–155 (2015)
8. Bernadet, A., Graham-Lengrand, S.: Non-idempotent intersection types and strong normalisation. Logical Methods in Computer Science **9**(4) (2013)
9. Bono, V., Venneri, B., Bettini, L.: A typed lambda calculus with intersection types. Theor. Comput. Sci. **398**(1-3), 95–113 (2008)
10. Bucciarelli, A., Ehrhard, T.: On phase semantics and denotational semantics: the exponentials. Ann. Pure Appl. Logic **109**(3), 205–241 (2001)
11. Bucciarelli, A., Ehrhard, T., Manzonetto, G.: A relational semantics for parallelism and non-determinism in a functional setting. Ann. Pure Appl. Logic **163**(7), 918–934 (2012)
12. Bucciarelli, A., Kesner, D., Ventura, D.: Non-idempotent intersection types for the lambda-calculus. Logic Journal of the IGPL **25**(4), 431–464 (2017)
13. Carraro, A., Guerrieri, G.: A Semantical and Operational Account of Call-by-Value Solvability. In: FOSSACS 2014. pp. 103–118 (2014)
14. de Carvalho, D.: Sémantiques de la logique linéaire et temps de calcul. Thèse de doctorat, Université Aix-Marseille II (2007)
15. de Carvalho, D.: The relational model is injective for multiplicative exponential linear logic. In: CSL 2016. pp. 41:1–41:19 (2016)
16. de Carvalho, D.: Execution time of λ-terms via denotational semantics and intersection types. Math. Str. in Comput. Sci. **28**(7), 1169–1203 (2018)
17. de Carvalho, D., Pagani, M., Tortora de Falco, L.: A semantic measure of the execution time in linear logic. Theor. Comput. Sci. **412**(20), 1884–1902 (2011)
18. de Carvalho, D., Tortora de Falco, L.: A semantic account of strong normalization in linear logic. Inf. Comput. **248**, 104–129 (2016)
19. Coppo, M., Dezani-Ciancaglini, M.: A new type assignment for λ-terms. Arch. Math. Log. **19**(1), 139–156 (1978)
20. Coppo, M., Dezani-Ciancaglini, M.: An extension of the basic functionality theory for the λ-calculus. Notre Dame Journal of Formal Logic **21**(4), 685–693 (1980)
21. Curien, P.L., Herbelin, H.: The duality of computation. In: ICFP. pp. 233–243 (2000)
22. Díaz-Caro, A., Manzonetto, G., Pagani, M.: Call-by-value non-determinism in a linear logic type discipline. In: LFCS 2013. pp. 164–178 (2013)
23. Ehrhard, T.: Collapsing non-idempotent intersection types. In: CSL. pp. 259–273 (2012)

24. Ehrhard, T., Guerrieri, G.: The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In: PPDP 2016. pp. 174–187. ACM (2016)
25. Fiore, M.P., Plotkin, G.D.: An axiomatization of computationally adequate domain theoretic models of FPC. In: LICS '94. pp. 92–102 (1994)
26. Gardner, P.: Discovering needed reductions using type theory. In: TACS '94. Lecture Notes in Computer Science, vol. 789, pp. 555–574. Springer (1994)
27. Girard, J.Y.: Linear Logic. Theoretical Computer Science **50**, 1–102 (1987)
28. Girard, J.Y.: Normal functors, power series and the λ-calculus. Annals of Pure and Applied Logic **37**, 129–177 (1988)
29. Grégoire, B., Leroy, X.: A compiled implementation of strong reduction. In: ICFP '02. pp. 235–246 (2002)
30. Guerrieri, G.: Head reduction and normalization in a call-by-value lambda-calculus. In: WPTE 2015. pp. 3–17 (2015)
31. Guerrieri, G.: Towards a semantic measure of the execution time in call-by-value lambda-calculus. Tech. rep. (2018), submitted to ITRS 2018
32. Guerrieri, G., Paolini, L., Ronchi Della Rocca, S.: Standardization of a Call-By-Value Lambda-Calculus. In: TLCA 2015. pp. 211–225 (2015)
33. Guerrieri, G., Paolini, L., Ronchi Della Rocca, S.: Standardization and conservativity of a refined call-by-value lambda-calculus. Logical Methods in Computer Science **13**(4) (2017)
34. Guerrieri, G., Pellissier, L., Tortora de Falco, L.: Computing Connected Proof(-Structure)s from their Taylor Expansion. In: FSCD 2016. pp. 20:1–20:18 (2016)
35. Honda, K., Yoshida, N.: Game-theoretic analysis of call-by-value computation. Theor. Comput. Sci. **221**(1-2), 393–456 (1999)
36. Kesner, D., Vial, P.: Types as resources for classical natural deduction. In: FSCD 2017. LIPIcs, vol. 84, pp. 24:1–24:17 (2017)
37. Kfoury, A.J.: A linearization of the lambda-calculus and consequences. J. Log. Comput. **10**(3), 411–436 (2000)
38. Krivine, J.L.: λ-calcul, types et modèles. Masson (1990)
39. Lassen, S.: Eager Normal Form Bisimulation. In: LICS 2005. pp. 345–354 (2005)
40. Mazza, D., Pellissier, L., Vial, P.: Polyadic approximations, fibrations and intersection types. PACMPL **2**, 6:1–6:28 (2018)
41. Neergaard, P.M., Mairson, H.G.: Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In: ICFP 2004. pp. 138–149 (2004)
42. Ong, C.L.: Quantitative semantics of the lambda calculus: Some generalisations of the relational model. In: LICS 2017. pp. 1–12 (2017)
43. Paolini, L., Piccolo, M., Ronchi Della Rocca, S.: Essential and relational models. Mathematical Structures in Computer Science **27**(5), 626–650 (2017)
44. Paolini, L., Ronchi Della Rocca, S.: Call-by-value Solvability. ITA **33**(6), 507–534 (1999)
45. Plotkin, G.D.: Call-by-Name, Call-by-Value and the lambda-Calculus. Theor. Comput. Sci. **1**(2), 125–159 (1975)
46. Pottinger, G.: A type assignment for the strongly normalizable λ-terms. In: To HB Curry: essays on combinatory logic, λ-calculus and formalism. pp. 561–577 (1980)
47. Pravato, A., Ronchi Della Rocca, S., Roversi, L.: The call-by-value λ-calculus: a semantic investigation. Math. Str. in Comput. Sci. **9**(5), 617–650 (1999)
48. Ronchi Della Rocca, S., Paolini, L.: The Parametric λ-Calculus. Springer (2004)
49. Ronchi Della Rocca, S., Roversi, L.: Intersection logic. In: CSL 2001. pp. 414–428 (2001)
50. Sieber, K.: Relating full abstraction results for different programming languages. In: FSTTCS 1990. pp. 373–387 (1990)