Tight Typings and Split Bounds

BENIAMINO ACCATTOLI, Inria & École Polytechnique, France
STÉPHANE GRAHAM-LENGRAND, CNRS, Inria & École Polytechnique, France
DELIA KESNER, CNRS and Université Paris-Diderot, France

Multi types—aka non-idempotent intersection types—have been used to obtain quantitative bounds on higher-order programs, as pioneered by de Carvalho. Notably, they bound at the same time the number of evaluation steps *and* the size of the result. Recent results show that the number of steps can be taken as a reasonable time complexity measure. At the same time, however, these results suggest that multi types provide quite lax complexity bounds, because the size of the result can be exponentially bigger than the number of steps.

Starting from this observation, we refine and generalise a technique introduced by Bernadet & Graham-Lengrand to provide *exact* bounds for the maximal strategy. Our typing judgements carry two counters, one measuring evaluation lengths and the other measuring result sizes. In order to emphasise the modularity of the approach, we provide exact bounds for four evaluation strategies, both in the λ-calculus (head, leftmost-outermost, and maximal evaluation) and in the linear substitution calculus (linear head evaluation).

Our work aims at both capturing the results in the literature and extending them with new outcomes. Concerning the literature, it unifies de Carvalho and Bernadet & Graham-Lengrand via a uniform technique and a complexity-based perspective. The two main novelties are exact split bounds for the leftmost strategy—the only known strategy that evaluates terms to full normal forms and provides a reasonable complexity measure—and the observation that the computing device hidden behind multi types is the notion of substitution at a distance, as implemented by the linear substitution calculus.

CCS Concepts: • **Software and its engineering** → **General programming languages**; • **Social and professional topics** → *History of programming languages*;

Additional Key Words and Phrases: lambda-calculus, type systems, cost models

ACM Reference Format:
Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2018. Tight Typings and Split Bounds. *Proc. ACM Program. Lang.* 2, ICFP, Article 94 (September 2018), 30 pages. https://doi.org/10.1145/3236789

1 INTRODUCTION

Type systems enforce properties of programs, such as termination, deadlock-freedom, or productivity. This paper studies a class of type systems for the λ-calculus that refines termination by providing exact bounds for evaluation lengths and normal forms.

Intersection types and multi types. One of the cornerstones of the theory of λ-calculus is that intersection types *characterise* termination: not only typed programs terminate, but all terminating programs are typable as well [Coppo and Dezani-Ciancaglini 1978, 1980; Krivine 1993; Pottinger

Authors' addresses: Beniamino Accattoli, LIX, Inria & École Polytechnique, France, beniamino.accattoli@inria.fr; Stéphane Graham-Lengrand, LIX, CNRS, Inria & École Polytechnique, France, graham-lengrand@lix.polytechnique.fr; Delia Kesner, IRIF, CNRS and Université Paris-Diderot, France, kesner@irif.fr.

1980]. In fact, the λ-calculus comes with different notions of evaluation (*e.g.* call-by-name, call-by-value, call-by-need, etc) to different notions of normal forms (head/weak/full, etc) and, accordingly, with different systems of intersection types.

Intersection types are a flexible tool and, even when one fixes a particular notion of evaluation and normal form, the type system can be formulated in various ways. A flavour that became quite convenient in the last 10 years is that of *non-idempotent* intersection types [de Carvalho 2007; Gardner 1994; Kfoury 2000; Neergaard and Mairson 2004] (a survey can be found in [Bucciarelli et al. 2017]), where the intersection $A \cap A$ is not equivalent to A. Non-idempotent intersection types are more informative than idempotent ones because they give rise to a *quantitative* approach, that allows counting resource consumption.

Non-idempotent intersections can be seen as multi-sets, which is why, to ease the language, we prefer to call them *multi types* rather than *non-idempotent intersection types*. Multi types have two main features:

(1) *Bounds on evaluation lengths*: they go beyond simply qualitative characterisations of termination, as typing derivations provide quantitative bounds on the length of evaluation (*i.e.* on the number of β-steps). Therefore, they give intensional insights on programs, and seem to provide a tool to reason about the complexity of programs.
(2) *Linear logic interpretation*: multi types are deeply linked to linear logic. The relational model [Bucciarelli and Ehrhard 2001; Girard 1988] of linear logic (often considered as a sort of canonical model of linear logic) is based on multi-sets, and multi types can be seen as a syntactic presentation of the relational model of the λ-calculus induced by the interpretation into linear logic.

These two facts together have a potential, fascinating consequence: they suggest that denotational semantics may provide abstract tools for complexity analyses, that are theoretically solid, being grounded on linear logic.

Various works in the literature explore the bounding power of multi types. Often, the bounding power is used *qualitatively*, *i.e.* without explicitely counting the number of steps, to characterise termination and / or the properties of the induced relational model. Indeed, multi types provide combinatorial proofs of termination that are simpler than those developed for (idempotent) intersection types (*e.g.* reducibility candidates). Several papers explore this approach under the call-by-name [Bucciarelli et al. 2012; Kesner and Ventura 2015; Kesner and Vial 2017; Ong 2017; Paolini et al. 2017] or the call-by-value [Carraro and Guerrieri 2014; Díaz-Caro et al. 2013; Ehrhard 2012] operational semantics, or both [Ehrhard and Guerrieri 2016]. Sometimes, precise *quantitative* bounds are provided instead, as in [Bernadet and Graham-Lengrand 2013b; de Carvalho 2007]. Multi types can also be used to provide characterisation of complexity classes [Benedetti and Ronchi Della Rocca 2016]. Other qualitative [de Carvalho 2016; Guerrieri et al. 2016] and quantitative [de Carvalho et al. 2011; de Carvalho and Tortora de Falco 2016] studies are also sometimes done in the more general context of linear logic, rather than in the λ-calculus.

Reasonable cost models. Usually, the quantitative works define a measure for typing derivations and show that the measure provides a bound on the length of evaluation sequences for typed terms. A criticism that could be raised against these results is, or rather was, that the number of β-steps of the bounded evaluation strategies might not be a reasonable cost model, that is, it might not be a reliable complexity measure. This is because no reasonable cost models for the λ-calculus were known at the time. But the understanding of cost models for the λ-calculus made significant progress in the last few years. Since the nineties, it is known that the number of steps for *weak* strategies (*i.e.* not reducing under abstraction) is a reasonable cost model [Blelloch and Greiner 1995], where *reasonable* means polynomially related to the cost model of Turing machines. It is only in 2014, that a solution for the general case has been obtained: the length of leftmost evaluation to normal form was shown to be a reasonable cost model in [Accattoli and Dal Lago 2016]. In this work we essentially update the study of the bounding power of multi types with the insights coming from the study of reasonable cost models. In particular, we provide new answers to the question of whether denotational semantics can really be used as an accurate tool for complexity analyses.

Size explosion and lax bounds. The study of cost models made clear that evaluation lengths are independent from the size of their results. The skepticism about taking the number of β-steps as a reliable complexity measure comes from the *size explosion problem*, that is, the fact that the size of terms can grow exponentially with respect to the number of β-steps. When λ-terms are used to encode decision procedures, the normal forms (encoding true or false) are of constant size, and therefore there is no size explosion issue. But when λ-terms are used to compute other normal forms than Boolean values, there are families of terms $\{t_n\}_{n\in\mathbb{N}}$ where t_n has size linear in n, it evaluates to normal form in n β-steps, and produces a result p_n of size $\Omega(2^n)$, *i.e.* exponential in n. Moreover, the size explosion problem is extremely robust, as there are families for which the size explosion is independent of the evaluation strategy. The difficulty in proving that the length of a given strategy provides a reasonable cost model lies precisely in the fact that one needs a compact representation of normal forms, to avoid to fully compute them (because they can be huge and it would be too expensive). A divulgative introduction to reasonable cost models and size explosion is [Accattoli 2018].

Now, multi typings do bound the number of β-steps of reasonable strategies, but these bounds are too generous since they bound at the same time the length of evaluations and the size of the normal forms. Therefore, even a notion of *minimal* typing (in the sense of being the smallest derivation) provides a bound that in some cases is exponentially worse than the number of β-steps.

Our observation is that the typings themselves are in fact much bigger than evaluation lengths, and so the widespread point of view for which multi types—and so the relational model of linear logic—faithfully capture evaluation lengths, or even the complexity, is misleading.

1.1 Contributions

The tightening technique. Our starting point is a technique introduced in a technical report by [Bernadet and Graham-Lengrand 2013a]. They study the case of strong normalisation, and present a multi type system where typing derivations of terms provide an *upper bound* on the number of β-steps to normal form. More interestingly, they show that every strongly normalising term admits a typing derivation that is sufficiently tight, where the obtained bound is *exactly* the length of the longest β-reduction path. This improved on previous results, *e.g.* [Bernadet and Graham-Lengrand 2013b; Bernadet and Lengrand 2011] where multi types provided the exact measure of longest evaluation paths *plus the size of the normal forms* which, as discussed above, can be exponentially bigger. Finally, they enrich the structure of base types so that, for those typing derivations providing the exact lengths, the type of a term gives the structure (and hence the size) of its normal form. This paper embraces this tightening technique, simplifying it with the use of *tight constants* for base types, and generalising it to a range of other evaluation strategies, described below.

It is natural to wonder how natural the tightening technique is—a malicious reader may indeed suspect that we are cooking up an ad-hoc way of measuring evaluation lengths, betraying the *linear-logic-in-disguise* spirit of multi types. To remove any doubt, we show that our tight typings are actually isomorphic to minimal multi typings without tight constants. Said differently, the tightening technique turns out to be a way of characterising minimal typings in the standard multi type framework (aka the relational model). Let us point out that, in the literature, there are characterisations of minimal typings (so-called principal typings) only for normal forms, and they extend to non-normal terms only *indirectly*, that is, by subject expansion of those for normal forms. Our approach, instead, provides a *direct* description, for any typable term.

Modular approach. We develop all our results by using a unique schema that modularly applies to different evaluation strategies. Our approach isolates the key concepts for the correctness and completeness of multi types, providing a powerful and modular technique, having at least two by-products. First, it reveals the relevance of *neutral terms* and of their properties with respect to types. Second, the concrete instantiations of the schema on four different cases always require subtle definitions, stressing the key conceptual properties of each case study.

Head and leftmost evaluation. Our first application of the tightening technique is to the head and leftmost evaluation strategies. The head case is the simplest possible one. The leftmost case is the natural iteration of the head one, and the only known strong strategy whose number of steps provides a reasonable cost model [Accattoli and Dal Lago 2016]. Multi types bounding the lengths of leftmost normalising terms have been also studied in [Kesner and Ventura 2014], but the exact number of steps taken by the leftmost strategy has not been measured via multi types before—therefore, this is a new result, as we now explain.

The study of the head and the leftmost strategies, at first sight, seems to be a minor reformulation of de Carvalho's results about measuring via multi types the length of executions of the Krivine abstract machine (shortened KAM)—implementing weak head evaluation—and of the iterated KAM—that implements leftmost evaluation [de Carvalho 2009]. The study of cost models is here enlightening: de Carvalho's iterated KAM does implement leftmost evaluation, but the overhead of the machine (that is counted by de Carvalho's measure) is exponential in the number of β-steps, while here we only measure the number of β-steps, thus providing a much more parsimonious (and yet reasonable) measure.

The work of de Carvalho, Pagani and Tortora de Falco [de Carvalho et al. 2011], using the relational model of linear logic to measure evaluation lengths in proof nets, is also closely related. They do not however split the bounds, that is, they do not have a way to measure separately the number of steps and the size of the normal form. Moreover, their notion of cut-elimination by levels does not correspond to leftmost evaluation.

Maximal evaluation. We also apply the technique to the *maximal strategy*, which takes the maximum number of steps to normal form, if any, and diverges otherwise. The maximal strategy has been bounded in [Bernadet and Lengrand 2011], and exactly measured in [Bernadet and Graham-Lengrand 2013a] via the idea of tightening, as described above. With respect to [Bernadet and Graham-Lengrand 2013a], our technical development is simpler. The differences are:

(1) *Uniformity with other strategies*: The typing system used in [Bernadet and Graham-Lengrand 2013a] for the maximal strategy has a special rule for typing a λ-abstraction whose bound variable does not appear in the body. This special case is due to the fact that the *empty* multi type is forbidden in the grammar of function types. Here, we align the type grammar with that used for other evaluation strategies, allowing the empty multi type, which in turn allows the typing rules for λ-abstractions to be the same as for head and leftmost evaluation. This is not only simpler, but it also contributes to making the whole approach more uniform across the different strategies that we treat in the paper. Following the head and leftmost evaluation cases, our completeness theorem for the maximal strategy bears quantitative information (about *e.g.* evaluation lengths), in contrast with [Bernadet and Graham-Lengrand 2013a].

(2) *Quantitative aspects of normal forms*: Bernadet and Graham-Lengrand encode the shape of normal forms into base types. We simplify this by only using two tight constants for base types. On the other hand, we decompose the actual size of a typing derivation as the sum of two quantities: the first one is shown to match the maximal evaluation *length* of the typed term, and the second one is shown to match the *size* of its normal form together with the size of all terms that are erased by the evaluation process. Identifying what the second quantity captures is a new contribution.
(3) *Neutral terms*: we emphasise the key role of neutral terms in the technical development by describing their specificities with respect to typing. This is not explicitly broached in [Bernadet and Graham-Lengrand 2013a].

Linear head evaluation. Last, we apply the tightening technique to *linear* head evaluation [Danos and Regnier 2004; Mascari and Pedicini 1994] (*lhd* for short), formulated in the linear substitution calculus (LSC) [Accattoli 2012; Accattoli et al. 2014], a λ-calculus with explicit substitutions that is strongly related to linear logic proof nets, and also a minor variation over a calculus by Milner [Milner 2007]. The literature contains a characterisation of *lhd*-normalisable terms [Kesner and Ventura 2014]. Moreover, [de Carvalho 2007] measures the executions of the KAM, a result that can also be interpreted as a measure of *lhd*-evaluation. What we show however is stronger, and somewhat unexpected.

To bound *lhd*-evaluation, in fact, we can strongly stand on the bounds obtained for head evaluation. More precisely, the result for the exact bounds for *head* evaluation takes only into account the number of abstraction and application typing rules. For *linear* head evaluation, instead, we simply need to count also the axioms, *i.e.* the rules typing variable occurrences, nothing else. It turns out that the length of a linear head evaluation plus the size of the linear head normal form is *exactly* the size of the tight typing.

Said differently, multi typings simply encode evaluations in the LSC. In particular, we do not have to adapt multi types to the LSC, as for instance de Carvalho does to deal with the KAM. It actually is the other way around. As they are, multi typings naturally measure evaluations in the LSC. To measure evaluations in the λ-calculus, instead, one has to forget the role of the axioms. The best way to stress it, probably, is that the LSC is the computing device behind multi types.

Most proofs are provided in the long version of this paper [Accattoli et al. 2018].

1.2 Other Related Works

Apart from the papers already cited, let us mention some other related works. A recent, general categorical framework to define intersection and multi type systems is in [Mazza et al. 2018].

While the inhabitation problem is undecidable for idempotent intersection types [Urzyczyn 1999], the quantitative aspects provided by multi types make it decidable [Bucciarelli et al. 2014]. Intersection type are also used in [Dudenhefner and Rehof 2017] to give a bounded dimensional description of λ-terms via a notion of *norm*, which is resource-aware and orthogonal to that of *rank*. It is proved that inhabitation in bounded dimension is decidable (EXPSPACE-complete) and subsumes decidability in rank 2 [Urzyczyn 2009].

Other works propose a more practical perspective on resource-aware analyses for functional programs. In particular, type-based techniques for automatically inferring bounds on higher-order functions have been developed, based on sized types [Avanzini and Lago 2017; Hughes et al. 1996; Portillo et al. 2002; Vasconcelos and Hammond 2004] or amortized analysis [Hoffmann and Hofmann 2010; Hofmann and Jost 2003; Jost et al. 2017]. This led to practical cost analysis tools like *Resource-Aware ML* [Hoffmann et al. 2012] (see raml.co). Intersection types have been used [Simões et al. 2007] to address the *size aliasing* problem of sized types, whereby cost analysis sometimes overapproximates cost to the point of losing all cost information [Portillo et al. 2002]. How our multi types could further refine the integration of intersection types with sized types is a direction for future work, as is the more general combination of our method with the type-based cost analysis techniques mentioned above.

2 A BIRD'S EYE VIEW

Our study is based on a schema that is repeated for different evaluation strategies, making most notions parametric in the strategy $\to_S$ under study. The following concepts constitute the main ingredients of our technique:

(1) *Strategy, together with the normal, neutral, and abs predicates*: there is a (deterministic) evaluation strategy $\to_S$ whose normal forms are characterised via two related predicates, $\mathtt{normal}_S(t)$ and $\mathtt{neutral}_S(t)$, the intended meaning of the second one is that t is S-normal and can never behave as an abstraction (that is, it does not create a redex when applied to an argument). We further parametrise also this last notion by using a predicate $\mathtt{abs}_S(t)$ identifying abstractions, because the definition of deterministic strategies requires some subterms to not be abstractions.

(2) *Typing derivations*: there is a multi types system which has three features:
   - *Tight constants*: there are two new type constants `neutral` and `abs`, and rules to introduce them. As their name suggests, the constants `neutral` and `abs` are used to type terms whose normal form is neutral or an abstraction, respectively.
   - *Tight derivations*: there is a notion of tight derivation that requires a special use of the constants.
   - *Indices*: typing judgements have the shape $\Gamma \vdash^{(b,r)} t : \tau$, where b and r are indices meant to count, respectively, the number of steps to normal form and the size of the normal form.

(3) *Sizes*: there is a notion of size of terms that depends on the strategy, noted $|t|_S$. Moreover, there is a notion of size of typing derivations $|\Phi|_S$ that also depends on the strategy / type system, that coincides with the sum of the indices associated to the last judgement of Φ.

(4) *Characterisation*: we prove that $\Gamma \vdash^{(b,r)} t : \tau$ is a tight typing relatively to $\to_S$ if and only if there exists an S normal term p such that $t \to_S^{b/2} p$ and $|p|_S = r$.

(5) *Proof technique*: the characterisation is obtained always through the same sequence of intermediate results. Correctness follows from the fact that all tight typings of normal forms precisely measure their size, a *substitution lemma* for typing derivations and *subject reduction*. Completeness follows from the fact that every normal form admits a tight typing, an *anti-substitution lemma* for typing derivations, and *subject expansion*.

(6) *Neutral terms*: we stress the relevance of neutral terms in normalisation proofs from a typing perspective. In particular, correctness theorems always rely on a lemma about them. Neutral terms are a common concept in the study of λ-calculus, playing a key role in, for instance, the reducibility candidate technique [Girard et al. 1989].

The proof schema is illustrated in the next section on two standard strategies, namely *head* and *leftmost-outermost evaluation*. It is then slightly adapted to deal with *maximal evaluation* in Sect. 5 and *linear head evaluation* in Sect. 6.

Evaluation systems. Each case study treated in the paper relies on the same properties of the strategy $\to_S$ and the related predicates $\mathtt{normal}_S(t)$, $\mathtt{neutral}_S(t)$, and $\mathtt{abs}_S(t)$, that we collect under the notion of *evaluation system*.

Definition 2.1 (Evaluation system). *Let $\mathcal{T}_S$ be a set of terms, $\to_S$ be a (deterministic) strategy and $\mathtt{normal}_S$, $\mathtt{neutral}_S$, and $\mathtt{abs}_S$ be predicates on $\mathcal{T}_S$. All together they form an* evaluation system *S if for all $t, p, p_1, p_2 \in \mathcal{T}_S$:*

Head normal forms

$$\frac{}{\mathtt{neutral}_{hd}(x)} \qquad \frac{\mathtt{neutral}_{hd}(t)}{\mathtt{neutral}_{hd}(tp)} \qquad \frac{\mathtt{neutral}_{hd}(t)}{\mathtt{normal}_{hd}(t)} \qquad \frac{\mathtt{normal}_{hd}(t)}{\mathtt{normal}_{hd}(\lambda y.t)}$$

Leftmost-outermost normal forms

$$\frac{}{\mathtt{neutral}_{lo}(x)} \qquad \frac{\mathtt{neutral}_{lo}(t) \quad \mathtt{normal}_{lo}(p)}{\mathtt{neutral}_{lo}(tp)} \qquad \frac{\mathtt{neutral}_{lo}(t)}{\mathtt{normal}_{lo}(t)} \qquad \frac{\mathtt{normal}_{lo}(t)}{\mathtt{normal}_{lo}(\lambda y.t)}$$

Fig. 1. Head and leftmost-outermost neutral and normal terms

Head evaluation

$$\frac{}{(\lambda x.u)q \to_{hd} u\{x{\leftarrow}q\}} \qquad \frac{t \to_{hd} p}{\lambda x.t \to_{hd} \lambda x.p} \qquad \frac{t \to_{hd} p \quad \neg\mathtt{abs}_{hd}(t)}{tu \to_{hd} pu}$$

Leftmost-outermost evaluation

$$\frac{}{(\lambda x.u)q \to_{lo} u\{x{\leftarrow}q\}} \qquad \frac{t \to_{lo} p}{\lambda x.t \to_{lo} \lambda x.p} \qquad \frac{t \to_{lo} p \quad \neg\mathtt{abs}_{lo}(t)}{tu \to_{lo} pu} \qquad \frac{\mathtt{neutral}_{lo}(u) \quad t \to_{lo} p}{ut \to_{lo} up}$$

Fig. 2. Head and leftmost-outermost strategies

(1) Determinism of $\to_S$ *: if $t \to_S p_1$ and $t \to_S p_2$ then $p_1 = p_2$.*
(2) Characterisation of S-normal terms*: t is $\to_S$ -normal if and only if* $\mathtt{normal}_S(t)$*.*
(3) Characterisation of S-neutral terms*:* $\mathtt{neutral}_S(t)$ *if and only if* $\mathtt{normal}_S(t)$ *and* $\neg\mathtt{abs}_S(t)$*.*

Given a strategy $\to_S$ we use $\to_S^k$ for its k^{th} iteration and $\to_S^*$ for its transitive closure.

3 HEAD AND LEFTMOST-OUTERMOST EVALUATION

In this section we consider two evaluation systems at once. The two strategies are the famous *head* and *leftmost-outermost evaluation*. We treat the two cases together to stress the modularity of our technique. The set of λ-terms Λ is given by ordinary λ-terms:

$$\lambda\text{-Terms} \qquad t, p \quad ::= \quad x \mid \lambda x.t \mid tp$$

Normal, neutral, and abs predicates. The predicates $\mathtt{normal}_{hd}$ and $\mathtt{normal}_{lo}$ defining head and leftmost-outermost (shortened LO in the text and *lo* in mathematical symbols) normal terms are in Fig. 1, and they are based on two auxiliary predicates defining neutral terms: $\mathtt{neutral}_{hd}$ and $\mathtt{neutral}_{lo}$—note that $\mathtt{neutral}_{lo}(t)$ implies $\mathtt{neutral}_{hd}(t)$. The predicates $\mathtt{abs}_{hd}(t)$ and $\mathtt{abs}_{lo}(t)$ are equal for the systems *hd* and *lo* and they are true simply when t is an abstraction.

Small-step semantics. The *head* and *leftmost-outermost* strategies $\to_{hd}$ and $\to_{lo}$ are both defined in Fig. 2. Note that these definitions rely on the predicates defining neutral terms and abstractions.

Proposition 3.1 (Head and LO evaluation systems). *Let $S \in \{hd, lo\}$. Then* $(\Lambda, \to_S, \mathtt{neutral}_S, \mathtt{normal}_S, \mathtt{abs}_S)$ *is an evaluation system.*

The proof is routine.

$$\frac{}{x : [\tau] \vdash^{(0,0)} x : \tau}\ \mathsf{ax}$$

$$\frac{\Gamma \vdash^{(b,r)} t : \tau}{\Gamma \setminus\!\!\setminus x \vdash^{(b+1,r)} \lambda x.t : \Gamma(x) \to \tau}\ \mathsf{fun}_b \qquad \frac{\Gamma \vdash^{(b,r)} t : \mathtt{tight} \quad \mathtt{tight}(\Gamma(x))}{\Gamma \setminus\!\!\setminus x \vdash^{(b,r+1)} \lambda x.t : \mathtt{abs}}\ \mathsf{fun}_r$$

$$\frac{\Gamma \vdash^{(b,r)} t : \mathtt{M} \to \tau \quad \Delta \vdash^{(b',r')} p : \mathtt{M}}{\Gamma \uplus \Delta \vdash^{(b+b'+1,r+r')} tp : \tau}\ \mathsf{app}_b \qquad \frac{\Gamma \vdash^{(b,r)} t : \mathtt{neutral}}{\Gamma \vdash^{(b,r+1)} tp : \mathtt{neutral}}\ \mathsf{app}_r^{hd}$$

$$\frac{(\Delta_i \vdash^{(b_i,r_i)} t : \tau_i)_{i \in I}}{\uplus_{i \in I} \Delta_i \vdash^{(+_{i \in I} b_i, +_{i \in I} r_i)} t : [\tau_i]_{i \in I}}\ \mathsf{many} \qquad \frac{\Gamma \vdash^{(b,r)} t : \mathtt{neutral} \quad \Delta \vdash^{(b',r')} p : \mathtt{tight}}{\Gamma \uplus \Delta \vdash^{(b+b',r+r'+1)} tp : \mathtt{neutral}}\ \mathsf{app}_r^{lo}$$

Fig. 3. Type system for head and LO evaluations

Sizes. The notions of *head size* $|t|_{hd}$ and *LO size* $|t|_{lo}$ of a term t are defined as follows—the difference is on applications:

$$\begin{array}{rlcrl}
\multicolumn{2}{c}{\textsc{Head size}} & & \multicolumn{2}{c}{\textsc{LO size}} \\
|x|_{hd} & := 0 & & |x|_{lo} & := 0 \\
|\lambda x.p|_{hd} & := |p|_{hd} + 1 & & |\lambda x.p|_{lo} & := |p|_{lo} + 1 \\
|pu|_{hd} & := |p|_{hd} + 1 & & |pu|_{lo} & := |p|_{lo} + |u|_{lo} + 1
\end{array}$$

Multi types. We define the following notions about types.

- *Multi types* are defined by the following grammar:

$$\begin{array}{rrcl}
\textsc{Tight constants} & \mathtt{tight} & ::= & \mathtt{neutral} \mid \mathtt{abs} \\
\textsc{Types} & \tau, \sigma & ::= & \mathtt{tight} \mid X \mid \mathtt{M} \to \tau \\
\textsc{Multi-sets} & \mathtt{M} & ::= & [\tau_i]_{i \in I} \ (I \text{ a finite set})
\end{array}$$

  where X ranges over a non-empty set of *atomic types* and $[\ldots]$ denotes the multi-set constructor.
- *Examples* of multisets: $[\tau, \tau, \sigma]$ is a multi-set containing two occurrences of τ and one occurrence of σ, and $[\,]$ is the empty multi-set.
- A *typing context* Γ is a map from variables to finite multisets $\mathtt{M}$ of types such that only finitely many variables are not mapped to the empty multi-set $[\,]$. We write $\mathtt{dom}(\Gamma)$ for the domain of Γ, *i.e.* the set $\{x \mid \Gamma(x) \neq [\,]\}$.
- *Tightness*: we use the notation $\mathtt{Tight}$ for $[\mathtt{tight}]_{i \in I}$ (I a finite set). Moreover, we write $\mathtt{tight}(\tau)$ if τ is of the form $\mathtt{tight}$, $\mathtt{tight}(\mathtt{M})$ if $\mathtt{M}$ is of the form $\mathtt{Tight}$, and $\mathtt{tight}(\Gamma)$ if $\mathtt{tight}(\Gamma(x))$ for all x, in which case we also say that Γ is *tight*.
- The *multi-set union* $\uplus$ is extended to typing contexts point-wise, *i.e.* $\Gamma \uplus \Delta$ maps each variable x to $\Gamma(x) \uplus \Delta(x)$. This notion is extended to several contexts as expected so that $\uplus_{i \in I} \Gamma_i$ denotes a finite union of contexts (when $I = \emptyset$ the notation is to be understood as the empty context). We write $\Gamma; x : \mathtt{M}$ for $\Gamma \uplus (x \mapsto \mathtt{M})$ only if $x \notin \mathtt{dom}(\Gamma)$. More generally, we write $\Gamma; \Delta$ if the intersection between the domains of Γ and Δ is empty.
- The *restricted* context Γ with respect to the variable x, written $\Gamma \setminus\!\!\setminus x$ is defined by $(\Gamma \setminus\!\!\setminus x)(x) := [\,]$ and $(\Gamma \setminus\!\!\setminus x)(y) := \Gamma(y)$ if $y \neq x$.

Typing systems. There are two typing systems, one for head and one for LO evaluation. Their typing rules are presented in Fig. 3, the head system *hd* contains all the rules except $\mathtt{app}_r^{lo}$, the LO system *lo* contains all the rules except $\mathtt{app}_r^{hd}$.

Roughly, the intuitions behind the typing rules are (please ignore the indices b and r for the time being):

- *Rules* $\mathtt{ax}$, $\mathtt{fun}_b$, *and* $\mathtt{app}_b$: this rules are essentially the traditional rules for multi types for head and LO evaluation (see *e.g.* [Bucciarelli et al. 2017]), modulo the presence of the indices.
- *Rule* $\mathtt{many}$: this is a structural rule allowing typing terms with a multi-set of types. In some presentations of multi types $\mathtt{many}$ is hardcoded in the right premise of the $\mathtt{app}_b$ rule (that requires a multi-set). For technical reasons, it is preferable to separate it from $\mathtt{app}_b$. Morally, it corresponds to the !-promotion rule in linear logic.
- *Rule* $\mathtt{fun}_r$: t has already been tightly typed, and all the types associated to x are also tight constants. Then $\lambda x.t$ receives the tight constant $\mathtt{abs}$ for abstractions. The consequence is that this abstraction can no longer be applied, because it has not an arrow type, and there are no rules to apply terms of type $\mathtt{abs}$. Therefore, the abstraction constructor cannot be consumed by evaluation and it ends up in the normal form of the term, that has the form $\lambda x.t'$.
- *Rule* $\mathtt{app}_r^{hd}$: t has already been tightly typed with $\mathtt{neutral}$ and so morally it head normalises to a term t' having neutral form $xu_1 \dots u_k$. The rule adds a further argument p that cannot be consumed by evaluation, because t will never become an abstraction. Therefore, p ends up in the head normal form $t'p$ of tp, that is still neutral—correctly, so that tp is also typed with $\mathtt{neutral}$. Note that there is no need to type p because head evaluation never enters into arguments.
- *Rule* $\mathtt{app}_r^{lo}$: similar to rule $\mathtt{app}_r^{hd}$, except that LO evaluation enters into arguments and so the added argument p now also has to be typed, and with a tight constant. Note a key difference with $\mathtt{app}_b$: in $\mathtt{app}_r^{lo}$ the argument p is typed exactly once (that is, the type is not a multi-set)—correctly, because its LO normal form p' appears exactly once in the LO normal form $t'p'$ of tp (where t' is the LO normal form of t).
- *Tight constants and predicates*: there is of course a correlation between the tight constants $\mathtt{neutral}$ and $\mathtt{abs}$ and the predicates $\mathtt{neutral}_S$ and $\mathtt{abs}_S$. Namely, a term t is S-typable with $\mathtt{neutral}$ if and only if the S-normal form of t verifies the predicate $\mathtt{neutral}_S$, as we shall prove. For the tight constant $\mathtt{abs}$ and the predicate $\mathtt{abs}_S$ the situation is similar but weaker: if the S-normal form of t verifies $\mathtt{abs}_S$ then t is typable with $\mathtt{abs}$, but not the other way around—for instance a variable is typable with $\mathtt{abs}$ without being an abstraction.
- The type systems are not syntax-directed, *e.g.* given an abstraction (resp. an application), it can be typed with rule $\mathtt{fun}_r$ or $\mathtt{fun}_b$ (resp. $\mathtt{app}_r$ or $\mathtt{app}_b$), depending on whether the constructor typed by the rule ends up in the normal form or not. Thus for example, given the term $\mathtt{II}$, where $\mathtt{I}$ is the identity function $\lambda z.z$, the second occurrence of $\mathtt{I}$ can be typed with $\mathtt{abs}$ using rule $\mathtt{fun}_r$, while the first one can be typed with $[\mathtt{abs}] \to \mathtt{abs}$ using rule $\mathtt{fun}_b$.

Typing judgements are of the form $\Gamma \vdash^{(b,r)} t : \tau$, where (b, r) is a pair of integers whose intended meaning is explained in the next paragraph. We write $\Phi \triangleright_S \Gamma \vdash^{(b,r)} t : \tau$, with S being either *hd* or *lo*, if Φ is a typing derivation in the system S and ends in the judgement $\Gamma \vdash^{(b,r)} t : \tau$.

Indices. The roles of b and r can be described as follows:

- *b and β-steps*: b counts the rules of the derivation that can be used to form β-redexes, *i.e.* the number of $\mathtt{fun}_b$ and $\mathtt{app}_b$ rules. Morally, b is at least twice the number of β-steps to normal form because typing a β-redex requires two rules. For tight typing derivations (introduced below), we are going to prove that b is the exact (double of the) length of the evaluation of the typed term to its normal form, according to the chosen evaluation strategy.

- *r and size of the result*: r counts the rules typing constructors that cannot be consumed by β-reduction according to the chosen evaluation strategy. It counts the number of $\mathtt{fun}_r$ and $\mathtt{app}_r$. These rules type the result of the evaluation, according to the chosen strategy, and measure the size of the result. Both the notion of result and the way its size is measured depend on the evaluation strategy.

Typing size. We define both the *head* and the *LO size* $|\Phi|_{hd}$ and $|\Phi|_{lo}$ of a typing derivation Φ as the number of rules in Φ, not counting rules ax and many. The size of a derivation is reflected by the pair of indices (b, r) on its final judgement: whenever $\Phi \triangleright_S \ \Gamma \vdash^{(b,r)} t : \tau$, we have $b + r = |\Phi|_S$. Note indeed that every rule (except ax and many) adds exactly 1 to this size.

For systems *hd* and *lo*, the indices on typing judgements are not really needed, as b can be recovered as the number of $\mathtt{fun}_b$ and $\mathtt{app}_b$ rules, and r as the number of $\mathtt{fun}_r$ and $\mathtt{app}_r^{hd}/\mathtt{app}_r^{lo}$ rules. We prefer to make them explicit because 1) we want to stress the separate counting, and 2) for linear head evaluation in Sect. 6 the counting shall be more involved, and the indices shall not be recoverable.

The fact that ax is not counted for $|\Phi|_{hd}$ and $|\Phi|_{lo}$ shall change in Sect. 6, where we show that counting ax rules corresponds to measure evaluations in the linear substitution calculus. The fact that many is not counted, instead, is due to the fact that it does not correspond to any constructor on terms. A further reason is that the rule may be eliminated by absorbing it in the $\mathtt{app}_b$ rule, that is the only rule that uses multi-sets—it is however technically convenient to separate the two.

Subtleties and easy facts. Let us overview some peculiarities and consequences of the definition of our type systems.

1. *Relevance*: No weakening is allowed in axioms. An easy induction on typing derivations shows that a variable declaration $x : \mathtt{M} \neq [\,]$ appears explicitly in the typing context Γ of a type derivation for t only if x occurs free in some *typed* subterm of t. In system *lo*, all subterms of t are typed, and so $x : \mathtt{M} \neq [\,]$ appears in Γ if and only if $x \in \mathtt{fv}(t)$. In system *hd*, instead, arguments of applications might not be typed (because of rule $\mathtt{app}_r^{hd}$), and so there may be $x \in \mathtt{fv}(t)$ but not appearing in Γ.
2. *Vacuous abstractions*: we rely on the convention that the two abstraction rules can always abstract a variable x not explicitly occurring in the context. Indeed, in the $\mathtt{fun}_b$ rule, if $x \notin \mathsf{dom}(\Gamma)$, then $\Gamma \setminus\!\!\setminus x$ is equal to Γ and $\Gamma(x)$ is $[\,]$, while in the $\mathtt{fun}_r$ rule, if $x \notin \mathsf{dom}(\Gamma)$, then $\Gamma(x)$ is $[\,]$ and thus $\mathtt{tight}([\,])$ holds.
3. *Head typings and applications*: note the $\mathtt{app}_r^{hd}$ rule types an application tp without typing the right subterm p. This matches the fact that tp is a head normal form when t is, independently of the status of p.

Tight derivations. A given term t may have many different typing derivations, indexed by different pairs (b, r). They always provide upper bounds on $\rightarrow_S$ -evaluation lengths and lower bounds on the S-size $|\cdot|_S$ of S-normal forms, respectively. The interesting aspect of our type systems, however, is that there is a simple description of a class of typing derivations that provide *exact* bounds for these quantities, as we shall show. Their definition relies on tight constants.

Definition 3.2 (Tight derivations).
Let $S \in \{hd, lo\}$. A derivation $\Phi \triangleright_S \ \Gamma \vdash^{(b,r)} t : \sigma$ is tight *if* $\mathtt{tight}(\sigma)$ *and* $\mathtt{tight}(\Gamma)$.

Let us stress that, remarkably, tightness is expressed as a property of the last judgement only. This is however not so unusual: characterisations of weakly normalising terms via intersection/multi types also rely on properties of the last judgement only, as discussed in Sect. 7.

In Sect. 7, in particular, we show the the size of a tight derivation for a term t is *minimal* among derivations for t. Moreover, it is also the same size of the minimal derivations making no use of tight constants nor rules using them. Therefore, tight derivations may be thought as a characterisation of minimal derivations.

Example. Let $t_0 = (\lambda x_1.(\lambda x_0.x_0x_1)x_1)\mathtt{I}$, where $\mathtt{I}$ is the identity function $\lambda z.z$. Let us first consider the head evaluation of t_0 to hd normal-form:

$$(\lambda x_1.(\lambda x_0.x_0x_1)x_1)\mathtt{I} \rightarrow_{hd} (\lambda x_0.x_0\mathtt{I})\mathtt{I} \rightarrow_{hd} \mathtt{I}\mathtt{I} \rightarrow_{hd} \mathtt{I}$$

The evaluation sequence has length 3. The head normal form has size 1. To give a tight typing for the term t_0 let us write abs_1 for $[\mathsf{abs}] \rightarrow \mathsf{abs}$. Then,

$$\dfrac{\dfrac{\dfrac{\dfrac{x_0 : [\mathsf{abs}_1] \vdash^{(0,0)} x_0 : \mathsf{abs}_1 \qquad \dfrac{x_1 : [\mathsf{abs}] \vdash^{(0,0)} x_1 : \mathsf{abs}}{x_1 : [\mathsf{abs}] \vdash^{(0,0)} x_1 : [\mathsf{abs}]}}{x_0 : [\mathsf{abs}_1], x_1 : [\mathsf{abs}] \vdash^{(1,0)} x_0x_1 : \mathsf{abs}}}{x_1 : [\mathsf{abs}] \vdash^{(2,0)} \lambda x_0.x_0x_1 : [\mathsf{abs}_1] \rightarrow \mathsf{abs}} \qquad \dfrac{x_1 : [\mathsf{abs}_1] \vdash^{(0,0)} x_1 : \mathsf{abs}_1}{x_1 : [\mathsf{abs}_1] \vdash^{(0,0)} x_1 : [\mathsf{abs}_1]}}{\dfrac{x_1 : [\mathsf{abs}, \mathsf{abs}_1] \vdash^{(3,0)} (\lambda x_0.x_0x_1)x_1 : \mathsf{abs}}{\vdash^{(4,0)} \lambda x_1.(\lambda x_0.x_0x_1)x_1 : [\mathsf{abs}, \mathsf{abs}_1] \rightarrow \mathsf{abs}}} \qquad \dfrac{\vdots}{\vdash^{(1,1)} \mathtt{I} : [\mathsf{abs}, \mathsf{abs}_1]}}{\vdash^{(6,1)} (\lambda x_1.(\lambda x_0.x_0x_1)x_1)\mathtt{I} : \mathsf{abs}}$$

Indeed, the pair $(6, 1)$ represents $6/2 = 3$ evaluation steps to hd normal-form and a head normal form of size 1.

3.1 Tight Correctness

Correctness of tight typings is the fact that whenever a term is *tightly* typable with indices (b, r), then b is exactly (the double of) the number of evaluation steps to S-normal form while r is exactly the size of the S-normal form. Thus, tight typing in system hd (resp. lo) gives information about hd-evaluation to hd-normal form (resp. lo-evaluation to lo-normal form). The correctness theorem is always obtained via three intermediate steps.

First step: tight typings of normal forms. The first step is to show that, when a tightly typed term is a S-normal form, then the first index b of its type derivation is 0, so that it correctly captures the (double of the) number of steps, and the second index r coincides exactly with its S-size.

Proposition 3.3 (Properties of hd and lo tight typings for normal forms). *Let $S \in \{hd, lo\}$, t be such that* $\mathtt{normal}_S(t)$*, and $\Phi \triangleright_S \Gamma \vdash^{(b,r)} t : \tau$ be a typing derivation.*

(1) Size bound*: $|t|_S \leq |\Phi|_S$.*
(2) Tightness*: if Φ is tight then $b = 0$ and $r = |t|_S$.*
(3) Neutrality*: if $\tau =$* $\mathtt{neutral}$ *then* $\mathtt{neutral}_S(t)$.

The proof is by induction on the typing derivation Φ. Let us stress three points:

(1) *Minimality*: the size of typings of a normal form t always bounds the size of t (Proposition 3.3.1), and therefore tight typings, that provide an exact bound (Proposition 3.3.2), are typing of minimal size. For the sake of conciseness, in most of the paper we focus on tight typings only. In Sect. 7, however, we study in detail the relationship between arbitrary typings and tight typings, extending their minimality beyond normal forms.
(2) *Size of tight typings*: note that Proposition 3.3.2 indirectly shows that all tight typings have the same indices, and therefore the same size. The only way in which two tight typings can differ, in fact, is whether the variables in the typing context are typed with $\mathtt{neutral}$ or abs,

but the structure of different typings is necessarily the same (which is also the structure of the S-normal form itself).

(3) *Unveiling of a key structural property*: Proposition 3.3 relies on the following interesting lemma about S-neutral terms and tight typings.

Lemma 3.4 (Tight spreading on neutral terms). *Let $S \in \{hd, lo\}$, t be such that* $\mathtt{neutral}_{hd}(t)$, *and $\Phi \triangleright_S \Gamma \vdash^{(b,r)} t : \tau$ be a typing derivation such that* $\mathtt{tight}(\Gamma)$. *Then* $\mathtt{tight}(\tau)$.

The lemma expresses the fact that tightness of neutral terms only depends on their contexts. Morally, this fact is what makes tightness to be expressible as a property of the final judgement only. We shall see in Sect. 7 that a similar property is hidden in more traditional approaches to weak normalisation (see Lemma 7.6). Such a spreading property appears repeatedly in our study, and we believe that its isolation is one of the contributions of our work, induced by the modular and comparative study of various strategies.

Second step: substitution lemma. Then one has to show that types, typings, and indices behave well with respect to substitution, which is essential, given that β-reduction is based on it.

Lemma 3.5 (Substitution and typings for *hd* and *lo*). *The following rule is admissible in both systems hd and lo:*

$$\frac{\Gamma \vdash^{(b,r)} p : \mathsf{M} \quad \Delta; x : \mathsf{M} \vdash^{(b',r')} t : \tau}{\Gamma \uplus \Delta \vdash^{(b+b',r+r')} t\{x \leftarrow p\} : \tau}\ \mathsf{subs}$$

Moreover if the derivations of the premisses are tight then so is the derivation of the conclusion.

The proof is by induction on the derivation of $\Delta; x : \mathsf{M} \vdash^{(b',r')} t : \tau$.

Note that the lemma also holds for $\mathsf{M} = [\,]$, in which case Γ is necessarily empty. In system *lo*, it is also true that if $\mathsf{M} = [\,]$ then $x \notin \mathtt{fv}(t)$ and $t\{x \leftarrow p\} = t$, because all free variables of t have non empty type in the typing context. As already pointed out, in system *hd* such a matching between free variables and typing contexts does not hold, and it can be that $\mathsf{M} = [\,]$ and yet $x \in \mathtt{fv}(t)$ and $t\{x \leftarrow p\} \neq t$.

Third step: quantitative subject reduction. Finally, one needs to shows a quantitative form of type preservation along evaluation. When the typing is tight, every evaluation step decreases the first index b of exactly 2 units, accounting for the application and abstraction constructor *consumed* by the firing of the redex.

Proposition 3.6 (Quantitative subject reduction for *hd* and *lo*). *Let $S \in \{hd, lo\}$. If $\Phi \triangleright_S \Gamma \vdash^{(b,r)} t : \tau$ is tight and $t \to_S p$ then $b \geq 2$ and there exists a tight typing Φ' such that $\Phi' \triangleright_S \Gamma \vdash^{(b-2,r)} p : \tau$.*

The proof is by induction on $t \to_S p$, and it relies on the substitution lemma (Lemma 3.5) for the base case of β-reduction at top level.

It is natural to wonder what happens when the typing is not tight. In the head case, the index b still decreases exactly of 2. In the lo case things are subtler—they are discussed in Sect. 7.

Summing up. The tight correctness theorem is proved by a straightforward induction on the evaluation length relying on quantitative subject reduction (Proposition 3.6) for the inductive case, and the properties of tight typings for normal forms (Proposition 3.3) for the base case.

Theorem 3.7 (Tight correctness for *hd* and *lo*). *Let $S \in \{hd, lo\}$ and $\Phi \triangleright_S \Gamma \vdash^{(b,r)} t : \tau$ be a tight derivation. Then there exists p such that $t \to_S^{b/2} p$,* $\mathtt{normal}_S(p)$, *and $|p|_S = r$. Moreover, if $\tau = \mathtt{neutral}$ then* $\mathtt{neutral}_S(p)$.

3.2 Tight Completeness

Completeness of tight typings (in system $S \in \{hd, lo\}$) expresses the fact that every S-normalising term has a tight derivation (in system S). As for correctness, the completeness theorem is always obtained via three intermediate steps, dual to those for correctness. Essentially, one shows that every normal form has a tight derivation and then extends the result to S-normalising term by pulling typability back through evaluation using a subject expansion property.

First step: normal forms are tightly typable. A simple induction on the structure of normal forms proves the following proposition.

Proposition 3.8 (Normal forms are tightly typable for hd and lo). *Let $S \in \{hd, lo\}$ and t be such that $\mathtt{normal}_S(t)$. Then there exists a tight derivation $\Phi \triangleright_S \Gamma \vdash^{(0,|t|_S)} t : \tau$. Moreover, if $\mathtt{neutral}_S(t)$ then $\tau = \mathtt{neutral}$, and if $\mathtt{abs}_S(t)$ then $\tau = \mathtt{abs}$.*

In contrast to the proposition for normal forms of the correctness part (Proposition 3.3), here there are no auxiliary lemmas, so the property is simpler.

Second step: anti-substitution lemma. In order to pull typability back along evaluation sequence, we have to first show that typability can also be pulled back along substitutions.

Lemma 3.9 (Anti-substitution and typings for hd and lo). *Let $S \in \{hd, lo\}$ and $\Phi \triangleright_S \Gamma \vdash^{(b,r)} t\{x \leftarrow p\} : \tau$. Then there exist:*

- *a multi-set M;*
- *a typing derivation $\Phi_t \triangleright_S \Gamma_t; x : \mathsf{M} \vdash^{(b_t, r_t)} t : \tau$; and*
- *a typing derivation $\Phi_p \triangleright_S \Gamma_p \vdash^{(b_p, r_p)} p : \mathsf{M}$*

such that:

- Typing context*: $\Gamma = \Gamma_t \uplus \Gamma_p$;*
- Indices*: $(b, r) = (b_t + b_p, r_t + r_p)$.*

Moreover, if Φ is tight then so are Φ_t and Φ_p.

The proof is by induction on Φ.

Let us point out that the anti-substitution lemma holds also in the degenerated case in which x does not occur in t and p is not S-normalising: rule many can indeed be used to type *any* term p with $\vdash^{(0,0)} p : [\,]$ by taking an empty set I of indices for the premises. Note also that this is *forced* by the fact that $x \notin \mathtt{fv}(t)$, and so $\Gamma_t(x) = [\,]$. Finally, this fact does not contradict the correctness theorem, because here p is typed with a multi-set, while the theorem requires a type.

Third step: quantitative subject expansion. This property guarantees that typability can be pulled back along evaluation sequences.

Proposition 3.10 (Quantitative subject expansion for hd and lo). *Let $S \in \{hd, lo\}$ and $\Phi \triangleright_S \Gamma \vdash^{(b,r)} p : \tau$ be a tight derivation. If $t \rightarrow_S p$ then there exists a (tight) typing Φ' such that $\Phi' \triangleright_S \Gamma \vdash^{(b+2,r)} t : \tau$.*

The proof is a simple induction over $t \rightarrow_S p$ using the anti-substitution lemma in the base case of evaluation at top level.

Summing up. The tight completeness theorem is proved by a straightforward induction on the evaluation length relying on quantitative subject expansion (Proposition 3.10) for the inductive case, and the existence of tight typings for normal forms (Proposition 3.8) for the base case.

Theorem 3.11 (Tight completeness for hd and lo). *Let $S \in \{hd, lo\}$ and $t \rightarrow_S^k p$ with $\mathtt{normal}_S(p)$. Then there exists a tight typing $\Phi \triangleright_S \Gamma \vdash^{(2k, |p|_S)} t : \tau$. Moreover, if $\mathtt{neutral}_S(p)$ then $\tau = \mathtt{neutral}$, and if $\mathtt{abs}_S(p)$ then $\tau = \mathtt{abs}$.*

4 EXTENSIONS AND DEEPER ANALYSES

In the rest of the paper we are going to further explore the properties of the tight approach to multi types along three independent axes:

(1) *Maximal evaluation*: we adapt the methodology to the case of maximal evaluation, which relates to strong normalisation in that the maximal evaluation strategy terminates only if the term being evaluated is strongly normalising. This case is a simplification of [Bernadet and Graham-Lengrand 2013a] that can be directly related to the head and leftmost evaluation cases. It is in fact very close to leftmost evaluation but for the fact that, during evaluation, typing contexts are not necessarily preserved and the size of the terms being erased has to be taken into account. The statements of the properties in Sections 3.1 and 3.2 have to be adapted accordingly.

(2) *Linear head evaluation*: we reconsider head evaluation in the linear substitution calculus obtaining exact bounds on the number of steps and on the size of normal forms. The surprise here is that the type system is essentially unchanged and that it is enough to count also axiom rules (that are ignored for head evaluation in the λ-calculus) in order to exactly bound also the number of *linear substitution* steps.

(3) *LO evaluation and minimal typings*: we explore the relationship between tight typings and traditional typings without tight constants. This study is done in the context of LO evaluation, that is the more relevant one with respect to cost models for the λ-calculus. We show in particular that tight typings are isomorphic to minimal traditional typings.

Let us stress that these three variations on a theme can be read independently.

5 MAXIMAL EVALUATION

In this section we consider the maximal strategy, which gives the longest evaluation sequence from any strongly normalising term to its normal form. The maximal evaluation strategy is *perpetual* in that, if a term t has a diverging evaluation path then the maximal strategy diverges on t. Therefore, its termination subsumes the termination of any other strategy, which is why it is often used to reason about the strong normalisation property [van Raamsdonk et al. 1999].

Strong normalisation and erasing steps. It is well-known that in the framework of relevant (*i.e.* without weakening) multi types it is technically harder to deal with strong normalisation (all evaluations terminate)—which is equivalent to the termination of the maximal strategy— than with weak normalisation (there is a terminating evaluation)—which is equivalent to the termination of the LO strategy. The reason is that one has to ensure that all subterms that are erased along any evaluation are themselves strongly normalising.

The simple proof technique that we used in the previous section does not scale up—in general—to strong normalisation (or to the maximal strategy), because subject reduction breaks for erasing steps, as they change the final typing judgement. Of course the same is true for subject expansion. There are at least three ways of circumventing this problem:

(1) *Memory*: to add a memory constructor, as in Klop's calculus [Klop 1980], that records the erased terms and allows evaluation inside the memory, so that diverging subterms are preserved. Subject reduction then is recovered.

(2) *Subsumption/weakening*: adding a simple form of sub-typing, that allows stabilising the final typing judgement in the case of an erasing step, or more generally, adding a strong form of weakening, that essentially removes the empty multi type.

(3) *Big-step subject reduction*: abandon the preservation of the typing judgement in the erasing cases, and rely on a more involved big-step subject reduction property relating the term

$$\frac{x \in \mathsf{fv}(u)}{(\lambda x.u)q \xrightarrow{0}_{mx} u\{x{\leftarrow}q\}} \qquad \frac{\mathsf{normal}_{mx}(q) \quad x \notin \mathsf{fv}(u)}{(\lambda x.u)q \xrightarrow{|q|_{mx}}_{mx} u} \qquad \frac{t \xrightarrow{r}_{mx} p \quad x \notin \mathsf{fv}(u)}{(\lambda x.u)t \xrightarrow{r}_{mx} (\lambda x.u)p}$$

$$\frac{t \xrightarrow{r}_{mx} p}{\lambda x.t \xrightarrow{r}_{mx} \lambda x.p} \qquad \frac{\neg\mathsf{abs}_{mx}(t) \quad t \xrightarrow{r}_{mx} p}{tu \xrightarrow{r}_{mx} pu} \qquad \frac{\mathsf{neutral}_{mx}(u) \quad t \xrightarrow{r}_{mx} p}{ut \xrightarrow{r}_{mx} up}$$

Fig. 4. Deterministic maximal strategy

Typing rules $\{\mathsf{ax}, \mathsf{fun}_b, \mathsf{fun}_r, \mathsf{app}_b, \mathsf{app}_r^{lo}\}$ plus

$$\frac{(\Delta_i \vdash^{(b_i,r_i)} t : \tau_i)_{i\in I} \quad |I| > 0}{+_{i\in I}\Delta_i \vdash^{(+_{i\in I} b_i, +_{i\in I} r_i)} t : [\tau_i]_{i\in I}}\ \mathsf{many}_{>0} \qquad \frac{\Delta \vdash^{(b,r)} t : \tau}{\Delta \vdash^{(b,r)} t : [\,]}\ \mathsf{none}$$

Fig. 5. Type system for maximal evaluation

directly to its normal form, stating in particular that the normal form is typable, potentially by a different type.

Surprisingly, the tight characterisation of the maximal strategy that we are going to develop does not need any of these workarounds: in the case of tight typings subject reduction for the maximal strategy holds, and the simple proof technique used before adapts smoothly. To be precise, an evaluation step may still change the final typing judgement, but the key point is that the judgement stays tight. Morally, we are employing a form of subsumption of tight contexts, but an extremely light one, that in particular does not require a sub-typing relation. We believe that this is a remarkable feature of tight multi types.

Maximal evaluation and predicates. The maximal strategy shares with LO evaluation the predicates $\mathsf{neutral}_{lo}$, normal_{lo}, abs_{lo}, and the notion of term size $|t|_{lo}$, which we respectively write $\mathsf{neutral}_{mx}$, normal_{mx}, abs_{mx}, and $|t|_{mx}$. We actually define, in Fig. 4, a version of the maximal strategy, denoted $\xrightarrow{r}_{mx}$, that is indexed by an integer r representing the size of what is erased by the evaluation step. We define the transitive closure of $\xrightarrow{r}_{mx}$ as follows:

$$\frac{}{t \xrightarrow{0}{}^{0}_{mx} t} \qquad \frac{t \xrightarrow{r_1}_{mx} p \quad p \xrightarrow{r_2}{}^{k}_{mx} u}{t \xrightarrow{r_1+r_2}{}^{k+1}_{mx} u} \qquad \frac{t \xrightarrow{r}{}^{k}_{mx} p}{t \xrightarrow{r}{}^{*}_{mx} p}$$

Proposition 5.1 (*mx* evaluation system). *$(\Lambda, \rightarrow_{mx}, \mathsf{neutral}_{mx}, \mathsf{normal}_{mx}, \mathsf{abs}_{mx})$ is an evaluation system.*

Also in this case the proof is routine.

Multi types. Multi types are defined exactly as in Section 3. The type system *mx* for *mx*-evaluation is defined in Fig. 5. Rules $\mathsf{many}_{>0}$ and none, which is a special 0-ary version of many, are used to prevent an argument p in rule app_b to be *untyped*: either it is typed by means of rule $\mathsf{many}_{>0}$—and thus it is typed with at least one type—or it is typed by means of rule none—and thus it is typed with exactly one type: the type itself is then forgotten, but requiring the premise to have a type forces the term to be normalising. The fact that arguments are always typed, even those that are erased during reduction, is essential to guarantee strong normalisation: system mx cannot type anymore a term like $x\Omega$. Note that if $\Phi \triangleright_{mx} \; \Gamma \vdash^{(b,r)} t : \tau$, then $x \in \mathtt{fv}(t)$ if and only if $\Gamma(x) \neq [\,]$.

Similarly to the head and leftmost-outermost cases, we define the *size* $|\Phi|_{mx}$ of a typing derivation Φ as the number of rule applications in Φ, not counting rules ax and $\mathsf{many}_{>0}$ and none. And again if $\Phi \triangleright_{mx} \; \Gamma \vdash^{(b,r)} t : \tau$ then $b + r = |\Phi|_{mx}$.

For maximal evaluation, we need also to refine the notion of tightness of typing derivations, which becomes a global condition because it is no longer a property of the final judgment only:

Definition 5.2 (Mx-tight derivations). *A derivation* $\Phi \triangleright_{mx} \; \Gamma \vdash^{(b,r)} t : \tau$ *is* garbage-tight *if in every instance of rule* (none) *in* Φ *we have* $\mathtt{tight}(\tau)$. *It is* mx-tight *if also* Φ *is tight, in the sense of Definition 3.2.*

Similarly to the head and LO cases, the quantitative information in mx-tight derivations characterises evaluation lengths and sizes of normal forms, as captured by the correctness and completeness theorems.

5.1 Tight Correctness

The correctness theorem is proved following the same schema used for head and LO evaluations. Most proofs are similar, and are therefore omitted.

We start with the properties of typed normal forms. As before, we need an auxiliary lemma about neutral terms, analogous to Proposition 3.3.

Lemma 5.3 (Tight spreading on neutral terms for mx). *If* $\mathtt{neutral}_{hd}(t)$ *and* $\Phi \triangleright_{mx} \Gamma \vdash^{(b,r)} t : \tau$ *such that* $\mathtt{tight}(\Gamma)$, *then* $\mathtt{tight}(\tau)$.

The general properties of typed normal forms hold as well.

Proposition 5.4 (Properties of mx-tight typings for normal forms). *Given* $\Phi \triangleright_{mx} \Gamma \vdash^{(b,r)} t : \tau$ *with* $\mathtt{normal}_{mx}(t)$,

(1) Size bound: $|t|_{mx} \leq b + r$.
(2) Tightness: *if* Φ *is mx-tight then* $b = 0$ *and* $r = |t|_{mx}$.
(3) Neutrality: *if* $\tau = \mathtt{neutral}$ *then* $\mathtt{neutral}_{mx}(t)$.

Then we can type substitutions:

Lemma 5.5 (Substitution and typings for mx). *The following rule is admissible in system mx:*

$$\frac{\Gamma \vdash^{(b,r)} p : \mathsf{M} \quad \Delta; x : \mathsf{M} \vdash^{(b',r')} t : \tau \quad \mathsf{M} \neq [\,]}{\Gamma \uplus \Delta \vdash^{(b+b',r+r')} t\{x \leftarrow p\} : \tau}\,\mathsf{subs}$$

Moreover if the derivations of the premisses are garbage-tight, then so is the derivation of the conclusion.

Note that, in contrast to Lemma 3.5 in Section 3.1, we assume that the multi-set M is not empty, so that the left premiss is derived with rule $\mathsf{many}_{>0}$ rather than none.

Subject reduction. The statement here slightly differs from the corresponding one in Section 3.1. Indeed, the typing environment Γ for term t is not necessarily preserved when typing p, because the evaluation step may erase a subterm. Consider for instance term $t = (\lambda x.x')(yy)$. In any mx-typing derivation of t, the typing context must declare y with an appropriate type that ensures that, when applying a well-typed substitution to t, the resulting term is still normalising for $\rightarrow_{mx}$. For instance, the context should declare $y : [[\tau] \rightarrow \tau, \tau]$, or even $y : [\mathtt{neutral}]$ if the typing derivation for t is mx-tight. However, as $t \xrightarrow{1}_{mx} x'$, the typing derivation for x' will clearly have a typing environment Γ' that maps y to $[\,]$. Hence, the subject reduction property has to take into account the change of typing context, as shown below.

Proposition 5.6 (Quantitative subject reduction for mx). *If $\Phi \triangleright_{mx} \Gamma \vdash^{(b,r)} t : \tau$ is mx-tight and $t \xrightarrow{e}_{mx} p$, then there exist Γ' and an mx-tight typing Φ' such that $\Phi' \triangleright_{mx} \Gamma' \vdash^{(b-2,r-e)} p : \tau$.*

Proof. See the long version of this paper [Accattoli et al. 2018]. □

Correctness theorem. Now the correctness theorem easily follows. It differs from the corresponding theorem in Section 3.1 in that the second index in the mx-tight typing judgement does not only measure the size of the normal form but also the sizes of all the terms erased during evaluation (and necessarily in normal form).

Theorem 5.7 (Tight correctness for mx-evaluation). *Let $\Phi \triangleright_{mx} \Gamma \vdash^{(b,r)} t : \tau$ be a mx-tight derivation. Then there is an integer e and a term p such that $\mathtt{normal}_{mx}(p)$, $t \overset{e}{\to}{}^{b/2}_{mx} p$ and $|p|_{mx} + e = r$. Moreover, if $\tau = \mathtt{neutral}$ then $\mathtt{neutral}_{mx}(p)$.*

5.2 Tight Completeness

Completeness is again similar to that in Section 3.2, and differs from it in the same way as correctness differs from that in Section 3.1. Namely, the second index in the completeness theorem also accounts for the size of erased terms, and the long version of this paper [Accattoli et al. 2018] provides the proof of the subject expansion property. The completeness statement follows.

Proposition 5.8 (Normal forms are tightly typable in mx). *Let t be such that $\mathtt{normal}_{mx}(t)$. Then there exists a mx-tight derivation $\Phi \triangleright_{mx} \Gamma \vdash^{(0,|t|_{mx})} t : \tau$. Moreover, if $\mathtt{neutral}_{mx}(t)$ then $\tau = \mathtt{neutral}$, and if $\mathtt{abs}_{mx}(t)$ then $\tau = \mathtt{abs}$.*

Lemma 5.9 (Anti-substitution and typings for mx). *If $\Phi \triangleright_{mx} \Gamma \vdash^{(b,r)} t\{x \leftarrow p\} : \tau$ and $x \in \mathtt{fv}(t)$, then there exist:*

- *a multiset $\mathtt{M}$ different from $[\,]$;*
- *a typing derivation $\Phi_t \triangleright_{mx} \Gamma_t; x : \mathtt{M} \vdash^{(b_t,r_t)} t : \tau$; and*
- *a typing derivation $\Phi_p \triangleright_{mx} \Gamma_p \vdash^{(b_p,r_p)} p : \mathtt{M}$*

such that:

- Typing context*: $\Gamma = \Gamma_t \uplus \Gamma_p$;*
- Indices*: $(b,r) = (b_t + b_p, r_t + r_p)$.*

Moreover, if Φ is garbage-tight then so are Φ_t and Φ_p.

Proposition 5.10 (Quantitative subject expansion for mx). *If $\Phi \triangleright_{mx} \Gamma \vdash^{(b,r)} p : \tau$ is mx-tight and $t \xrightarrow{e}_{mx} p$, then there exist Γ' and an mx-tight typing Φ' such that $\Phi' \triangleright_{mx} \Gamma' \vdash^{(b+2,r+e)} t : \tau$.*

Proof. See the long version of this paper [Accattoli et al. 2018]. □

Theorem 5.11 (Tight completeness for for mx). *If $t \overset{e}{\to}{}^{k}_{mx} p$ with $\mathtt{normal}_{mx}(p)$, then there exists an mx-tight typing $\Phi \triangleright_{mx} \Gamma \vdash^{(2k,|p|_{mx}+e)} t : \tau$. Moreover, if $\mathtt{neutral}_{mx}(p)$ then $\tau = \mathtt{neutral}$, and if $\mathtt{abs}_{mx}(p)$ then $\tau = \mathtt{abs}$.*

6 LINEAR HEAD EVALUATION

In this section we consider the linear version of the head evaluation system, where *linear* comes from the *linear substitution calculus* (LSC) [Accattoli 2012; Accattoli et al. 2014], a refinement of the λ-calculus where the language is extended with an explicit substitution constructor $t[x \backslash p]$, and *linear substitution* is a micro-step rewriting rule replacing one occurrence at a time—therefore, *linear* does not mean that variables have at most one occurrence, only that their occurrences are replaced one by one. Linear head evaluation—first studied in [Danos and Regnier 2004; Mascari and

Pedicini 1994]—admits various presentations. The one in the LSC adopted here has been introduced in [Accattoli 2012] and is the simplest one.

The insight here is that switching from head to linear head, and from the λ-calculus to the LSC only requires counting ax rules for the size of typings and the head variable for the size of terms—the type system, in particular is the same. The correspondence between the two system is spelled out in the last subsection of this part. Of course, switching to the LSC some details have to be adapted: a further index traces linear substitution steps, there is a new typing rule to type the new explicit substitution constructor, and the proof schema slightly changes, as the (anti-)substitution lemma is replaced by a partial substitution one—these are unavoidable and yet inessential modifications.

Thus, the main point of this section is to split the complexity measure among the multiplicative steps (beta steps) and the exponential ones (substitutions). Moreover, linear logic proof-nets are known to simulate the λ-calculus, and LSC is known to be isomorphic to the proof-nets used in the simulation. Therefore, the results of this section directly apply to those proof-nets.

Explicit substitutions. We start by introducing the syntax of our language, which is given by the following set $\Lambda_{\mathtt{lsc}}$ of terms, where $t[x\backslash p]$ is a new constructor called *explicit substitution* (shortened ES), that is equivalent to $\mathtt{let}\ x = p\ \mathtt{in}\ t$:

$$\text{LSC TERMS} \qquad t, p \quad ::= \quad x \mid \lambda x.t \mid tp \mid t[x\backslash p]$$

The notion of *free* variable is defined as expected, in particular, $\mathtt{fv}(t[x\backslash p]) := (\mathtt{fv}(t) \setminus \{x\}) \cup \mathtt{fv}(p)$. *(List of) substitutions* and *linear head contexts* are given by the following grammars:

$$\begin{array}{rrcl} \text{(LIST OF) SUBSTITUTION CONTEXTS} & \mathtt{L} & ::= & \langle\cdot\rangle \mid \mathtt{L}[x\backslash t] \\ \text{LINEAR HEAD CONTEXTS} & \mathtt{H} & ::= & \langle\cdot\rangle \mid \lambda x.\mathtt{H} \mid \mathtt{H}t \mid \mathtt{H}[x\backslash t] \end{array}$$

We write $\mathtt{L}\langle t\rangle$ (resp. $\mathtt{H}\langle t\rangle$) for the term obtained by replacing the whole $\langle\cdot\rangle$ in context $\mathtt{L}$ (resp. $\mathtt{H}$) by the term t. This *plugging* operation, as usual with contexts, can capture variables. We write $\mathtt{H}\langle\langle t\rangle\rangle$ when we want to stress that the context $\mathtt{H}$ does not capture the free variables of t.

Normal, neutral, and abs predicates. The predicate $\mathtt{normal}_{lhd}$ defining linear head normal terms and $\mathtt{neutral}_{lhd}$ defining linear head neutral terms are introduced in Fig. 6. They are a bit more involved than before, because switching to the micro-step granularity of the LSC the study of normal forms requires a finer analysis. The predicates are now based on three auxiliary predicates $\mathtt{neutral}^{x}_{lhd}$, $\mathtt{normal}^{x}_{lhd}$, and $\mathtt{normal}^{\#}_{lhd}$: the first two characterise neutral and normal terms whose head variable x is free, the third instead characterises normal forms whose head variable is bound. Note also that the abstraction predicate $\mathtt{abs}_{lhd}$ is now defined *modulo* ES, that is, a term such as $(\lambda x.t)[z\backslash p][y\backslash u]$ satisfies the predicate. It is worth noticing that a term t of the form $\mathtt{H}\langle\langle y\rangle\rangle$ does not necessarily verify $\mathtt{normal}_{lhd}(t)$, *e.g.* $(\lambda z.(yx)[x\backslash y])p$. Examples of linear head normal forms are $\lambda x.xy$ and $(yx)[x\backslash z](II)$.

Small-step semantics. Linear head evaluation is often specified by means of a non-deterministic strategy (having the diamond property) [Accattoli 2012]. Here, however, we present a minor deterministic variant, in order to follow the general schema presented in the introduction. The deterministic notion of linear head evaluation lhd is given in Fig. 7. An example of $\rightarrow_{lhd}$-sequence is

$$\begin{array}{llll} ((\lambda z.(xx)[x\backslash y])p)[y\backslash w] & \rightarrow_{lhd} & (xx)[x\backslash y][z\backslash p][y\backslash w] & \rightarrow_{lhd} \\ (yx)[x\backslash y][z\backslash p][y\backslash w] & \rightarrow_{lhd} & (wx)[x\backslash y][z\backslash p][y\backslash w] & \end{array}$$

From now on, we split the evaluation relation $\rightarrow_{lhd}$ in two different relations, *multiplicative* $\rightarrow_{\mathtt{m}}$ and *exponential* $\rightarrow_{\mathtt{e}}$ evaluation, where $\rightarrow_{\mathtt{m}}$ (resp. $\rightarrow_{\mathtt{e}}$) is generated by the base case $(lhd_{\mathtt{m}})$ (resp. $(lhd_{\mathtt{e}})$) and closed by the three rules (lhd_{λ}), $(lhd_{@})$, (lhd_{s}). The terminology *multiplicative* and *exponential* comes from the linear logic interpretation of the LSC. The literature contains also an

LINEAR HEAD ABSTRACTIONS

$$\frac{}{\mathsf{abs}_{lhd}(\lambda x.t)} \qquad \frac{\mathsf{abs}_{lhd}(t)}{\mathsf{abs}_{lhd}(t[x\backslash p])}$$

LINEAR HEAD NORMAL FORMS

$$\frac{}{\mathsf{neutral}^{x}_{lhd}(x)} \qquad \frac{\mathsf{neutral}^{x}_{lhd}(t)}{\mathsf{neutral}^{x}_{lhd}(tp)} \qquad \frac{\mathsf{neutral}^{x}_{lhd}(t) \quad y \neq x}{\mathsf{neutral}^{x}_{lhd}(t[y\backslash p])}$$

$$\frac{\mathsf{neutral}^{x}_{lhd}(t)}{\mathsf{normal}^{x}_{lhd}(t)} \qquad \frac{\mathsf{normal}^{x}_{lhd}(t) \quad y \neq x}{\mathsf{normal}^{x}_{lhd}(\lambda y.t)} \qquad \frac{\mathsf{normal}^{x}_{lhd}(t) \quad y \neq x}{\mathsf{normal}^{x}_{lhd}(t[y\backslash p])}$$

$$\frac{\mathsf{normal}^{x}_{lhd}(t)}{\mathsf{normal}^{\#}_{lhd}(\lambda x.t)} \qquad \frac{\mathsf{normal}^{\#}_{lhd}(t)}{\mathsf{normal}^{\#}_{lhd}(\lambda x.t)} \qquad \frac{\mathsf{normal}^{\#}_{lhd}(t)}{\mathsf{normal}^{\#}_{lhd}(t[y\backslash p])}$$

$$\frac{\mathsf{normal}^{x}_{lhd}(t)}{\mathsf{normal}_{lhd}(t)} \qquad \frac{\mathsf{normal}^{\#}_{lhd}(t)}{\mathsf{normal}_{lhd}(t)} \qquad \frac{\mathsf{neutral}^{x}_{lhd}(t)}{\mathsf{neutral}_{lhd}(t)}$$

Fig. 6. linear head neutral and normal forms

$$\frac{}{\mathsf{L}\langle \lambda x.t\rangle u \rightarrow_{lhd} \mathsf{L}\langle t[x\backslash u]\rangle}\ (lhd_{\mathsf{m}}) \qquad \frac{}{\mathsf{H}\langle\!\langle x\rangle\!\rangle[x\backslash t] \rightarrow_{lhd} \mathsf{H}\langle\!\langle t\rangle\!\rangle[x\backslash t]}\ (lhd_{\mathsf{e}})$$

$$\frac{t \rightarrow_{lhd} u}{\lambda x.t \rightarrow_{lhd} \lambda x.u}\ (lhd_{\lambda}) \qquad \frac{t \rightarrow_{lhd} u \quad \neg\mathsf{abs}_{lhd}(t)}{tv \rightarrow_{lhd} uv}\ (lhd_{@}) \qquad \frac{t \rightarrow_{lhd} u \quad t \neq \mathsf{H}\langle\!\langle x\rangle\!\rangle}{t[x\backslash v] \rightarrow_{lhd} u[x\backslash v]}\ (lhd_{s})$$

Fig. 7. Deterministic linear-head evaluation

alternative terminology, using B *at a distance* for $\rightarrow_{\mathsf{m}}$ (or *distant* B, where B is a common name for the variant of β introducing an ES instead of using meta-level substitution) and *linear substitution* for $\rightarrow_{\mathsf{e}}$.

PROPOSITION 6.1 (LINEAR HEAD EVALUATION SYSTEM).
$(\Lambda_{\mathsf{lsc}}, \rightarrow_{lhd}, \mathsf{neutral}_{lhd}, \mathsf{normal}_{lhd}, \mathsf{abs}_{lhd})$ *is an evaluation system.*

In the linear case the proof is subtler than for the head, LO, and maximal cases. It can be found in the long version of this paper [Accattoli et al. 2018].

Sizes. The notion of *linear head size* $|t|_{lhd}$ extends the head size to terms with ES by counting 1 for variables—note that ES do not contribute to the linear head size:

LINEAR HEAD SIZE

$$\begin{array}{lcllcl} |x|_{lhd} & := & 1 & |\lambda x.t|_{lhd} & := & |t|_{lhd} + 1 \\ |tu|_{lhd} & := & |t|_{lhd} + 1 & |t[x\backslash u]|_{lhd} & := & |t|_{lhd} \end{array}$$

Multi types. We consider the same multi types of Sect. 3, but now typing judgements are of the form $\Gamma \vdash^{(b,e,r)} t : \tau$, where (b, e, r) is a *triple* of integers whose intended meaning is explained in the

$$\frac{}{x : [\tau] \vdash^{(0,0,1)} x : \tau}\ \mathsf{ax} \qquad \frac{(\Delta_i \vdash^{(b_i,e_i,r_i)} t : \tau_i)_{i\in I}}{\uplus_{i\in I}\Delta_i \vdash^{(+_{i\in I} b_i, +e_i, +_{i\in I} r_i)} t : [\tau_i]_{i\in I}}\ \mathsf{many}$$

$$\frac{\Gamma; x : \mathtt{M} \vdash^{(b,e,r)} t : \tau}{\Gamma \vdash^{(b+1, e+|\mathtt{M}|, r-|\mathtt{M}|)} \lambda x.t : \mathtt{M} \to \tau}\ \mathtt{fun}_b \qquad \frac{\Gamma; x : \mathtt{Tight} \vdash^{(b,e,r)} t : \mathtt{tight}}{\Gamma \vdash^{(b,e,r+1)} \lambda x.t : \mathtt{abs}}\ \mathtt{fun}_r$$

$$\frac{\Gamma \vdash^{(b,e,r)} t : \mathtt{M} \to \tau \qquad \Delta \vdash^{(b',e',r')} p : \mathtt{M}}{\Gamma \uplus_{i\in I} \Delta_i \vdash^{(b+b'+1, e+e', r+r')} tp : \tau}\ \mathtt{app}_b \qquad \frac{\Gamma \vdash^{(b,e,r)} t : \mathtt{neutral}}{\Gamma \vdash^{(b,e,r+1)} tu : \mathtt{neutral}}\ \mathtt{app}_r$$

$$\frac{\Gamma; x : \mathtt{M} \vdash^{(b,e,r)} t : \tau \qquad \Delta \vdash^{(b',e',r')} u : \mathtt{M}}{\Gamma \uplus \Delta \vdash^{(b+b', e+e'+|\mathtt{M}|, r+r'-|\mathtt{M}|)} t[x\backslash u] : \tau}\ \mathsf{ES}$$

Fig. 8. Type system for linear head evaluation.

next paragraph. The typing system *lhd* is defined in Fig. 8. By abuse of notation, we use for all the typing rules—except ES which is a new rule—the same names used for *hd*.

The *linear head size* $|\Phi|_{lhd}$ of a typing derivation Φ is the number of rules in Φ, without counting the occurrences of many.

Note that $\Gamma \vdash^{(b,e,r)} \mathtt{H}\langle\!\langle x \rangle\!\rangle : \tau$ implies that $\Gamma = \Gamma'; x : \mathtt{M}$ with $\mathtt{M} \neq [\,]$.

Indices. The roles of the three components of (b, e, r) in a typing derivation $\Gamma \vdash^{(b,e,r)} t : \tau$ can be described as follows:

- *b and multiplicative steps*: b counts the rules of the derivation that can be used to form multiplicative redexes, *i.e.* subterms of the form $\mathtt{L}\langle \lambda x.t\rangle u$. To count this kind of redexes it is necessary to count the number of $\mathtt{fun}_b$ and $\mathtt{app}_b$ rules. As in the case of head evaluation (Sect. 3), b is at least twice the number of multiplicative steps to normal form because typing such redexes requires (at least) these two rules.
- *e and exponential steps*: e counts the rules of the derivation that can be used to form exponential redexes, *i.e.* subterms of the form $\mathtt{H}\langle\!\langle x \rangle\!\rangle[x\backslash t]$. To count this kind of redexes it is necessary to count, for every variable x, the number of occurrences that can be substituted during evaluation. The ES is not counted because a single ES can be involved in many exponential steps along an evaluation sequence.
- *r and size of the result*: r counts the rules typing variables, abstractions and applications (*i.e.* ax, $\mathtt{fun}_r$ and $\mathtt{app}_r$) that cannot be consumed by *lhd* evaluation, so that they appear in the linear head normal form of a term. Note that the ES constructor is not consider part of the head of terms.

Note also that the typing rules assume that variable occurrences (corresponding to ax rules) end up in the result, by having the third index set to 1. When a variable x becomes bound by an ES (rule ES) or by an abstraction destined to be applied ($\mathtt{fun}_b$), the number of uses of x, expressed by the multiplicity of the multi-set $\mathtt{M}$ typing it, is subtracted from the size of the result, because those uses of x correspond to the times that it shall be replaced via a linear substitution step, and thus they should no longer be considered as contributing to the result. Coherently, that number instead contributes to the index tracing linear substitution steps.

Definition 6.2 (Tight derivations). *A derivation* $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t : \sigma$ *is* tight *if* $\mathtt{tight}(\sigma)$ *and* $\mathtt{tight}(\Gamma)$.

Example. Let us give a concret example in system lhd. Consider again the term $t_0 = (\lambda x_1.(\lambda x_0.x_0x_1)x_1)\mathtt{I}$, where $\mathtt{I}$ is the identity function $\lambda z.z$. The linear head evaluation sequence from t_0 to lhd normal-form is given below, in which we distinguish the multiplicative steps from the exponential ones.

$$\begin{array}{llll}
(\lambda x_1.(\lambda x_0.x_0x_1)x_1)\mathtt{I} & \rightarrow_{\mathtt{m}} & ((\lambda x_0.x_0x_1)x_1)[x_1\backslash\mathtt{I}] & \rightarrow_{\mathtt{m}} \\
(x_0x_1)[x_0\backslash x_1][x_1\backslash\mathtt{I}] & \rightarrow_{\mathtt{e}} & (x_1x_1)[x_0\backslash x_1][x_1\backslash\mathtt{I}] & \rightarrow_{\mathtt{e}} \\
(\mathtt{I}x_1)[x_0\backslash x_1][x_1\backslash\mathtt{I}] & \rightarrow_{\mathtt{m}} & x_3[x_3\backslash x_1][x_0\backslash x_1][x_1\backslash\mathtt{I}] & \rightarrow_{\mathtt{e}} \\
x_1[x_3\backslash x_1][x_0\backslash x_1][x_1\backslash\mathtt{I}] & \rightarrow_{\mathtt{e}} & \mathtt{I}[x_3\backslash x_1][x_0\backslash x_1][x_1\backslash\mathtt{I}] &
\end{array}$$

The evaluation sequence has length 7: 3 multiplicative steps and 4 exponential steps. The linear head normal form has size 2. We now give a tight typing for the term t_0, by writing again abs_1 for $[\mathsf{abs}] \rightarrow \mathsf{abs}$.

$$\begin{prooftree}
\AxiomC{$x_0 : [\mathsf{abs}_1] \vdash^{(0,0,1)} x_0 : \mathsf{abs}_1$}
\AxiomC{$x_1 : [\mathsf{abs}] \vdash^{(0,0,1)} x_1 : \mathsf{abs}$}
\UnaryInfC{$x_1 : [\mathsf{abs}] \vdash^{(0,0,1)} x_1 : [\mathsf{abs}]$}
\BinaryInfC{$x_0 : [\mathsf{abs}_1], x_1 : [\mathsf{abs}] \vdash^{(1,0,2)} x_0x_1 : \mathsf{abs}$}
\UnaryInfC{$x_1 : [\mathsf{abs}] \vdash^{(2,1,1)} \lambda x_0.x_0x_1 : [\mathsf{abs}_1] \rightarrow \mathsf{abs}$}
\AxiomC{$x_1 : [\mathsf{abs}_1] \vdash^{(0,0,1)} x_1 : \mathsf{abs}_1$}
\UnaryInfC{$x_1 : [\mathsf{abs}_1] \vdash^{(0,0,1)} x_1 : [\mathsf{abs}_1]$}
\BinaryInfC{$x_1 : [\mathsf{abs}, \mathsf{abs}_1] \vdash^{(3,1,2)} (\lambda x_0.x_0x_1)x_1 : \mathsf{abs}$}
\UnaryInfC{$\vdash^{(4,3,0)} \lambda x_1.(\lambda x_0.x_0x_1)x_1 : [\mathsf{abs}, \mathsf{abs}_1] \rightarrow \mathsf{abs}$}
\AxiomC{$\vdots$}
\UnaryInfC{$\vdash^{(1,1,2)} \mathtt{I} : [\mathsf{abs}, \mathsf{abs}_1]$}
\BinaryInfC{$\vdash^{(6,4,2)} (\lambda x_1.(\lambda x_0.x_0x_1)x_1)\mathtt{I} : \mathsf{abs}$}
\end{prooftree}$$

Indeed, the pair $(6, 4, 2)$ represents $6/2 = 3$ (resp. 4) multiplicative (resp. exponential) evaluation steps to lhd normal-form, and a linear head normal form of size 2.

6.1 Tight Correctness

As in the case of head and LO evaluation, the correctness proof is based on three main properties: properties of normal forms—themselves based on a lemma about neutral terms—the interaction between (linear head) substitution and typings, and subject reduction.

Neutral terms ad properties of normal forms. As for the head case, the properties of tight typing of lhd normal forms depend on a spreading property of lhd neutral terms. Additionally, they also need to characterise the shape of typing contexts for tight typings of neutral and normal terms.

Lemma 6.3 (Tight spreading on neutral terms, plus typing contexts). *Let $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t : \tau$ be a derivation.*

(1) If $\mathtt{neutral}^{x}_{lhd}(t)$ then $x \in \mathsf{dom}(\Gamma)$. Moreover, if $\Gamma(x) = \mathtt{Tight}$ then $\tau = \mathtt{tight}$ and $\mathsf{dom}(\Gamma) = \{x\}$.

(2) If $\mathtt{normal}^{x}_{lhd}(t)$ then $x \in \mathsf{dom}(\Gamma)$. Moreover, if $\Gamma(x) = \mathtt{Tight}$ then $\mathsf{dom}(\Gamma) = \{x\}$.

(3) If $\mathtt{normal}^{\#}_{lhd}(t)$ and $\tau = \mathtt{tight}$ then $\tau = \mathsf{abs}$ and Γ is empty.

Proposition 6.4 (Properties of lhd tight typings for normal forms). *Let t be such that $\mathtt{normal}_{lhd}(t)$, and $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t : \tau$ be a typing derivation.*

(1) Size bound*: $|t|_{lhd} \leq |\Phi|_{lhd}$.*

(2) Tightness*: if Φ is tight then $b = e = 0$ and $r = |t|_{lhd}$.*

(3) Neutrality*: if $\tau = \mathtt{neutral}$ then $\mathtt{neutral}_{lhd}(t)$.*

Partial substitution lemma. The main difference in the proof schema with respect to the head case is about the substitution lemma, that is now expressed differently, because evaluation no longer relies on meta-level substitution. Linear substitutions consume one type at a time: performing a linear head substitution on a term of the form $\mathtt{H}\langle\!\langle x \rangle\!\rangle[x\backslash t]$ consumes *exactly* one type resource associated to the variable x, and all the other ones remain in the typing context after the partial substitution. Formally, an induction on $\mathtt{H}$ is used to show:

Lemma 6.5 (Partial substitution and typings for *lhd*). *The following rule is admissible in system lhd:*

$$\frac{x : \mathsf{M}; \Gamma \vdash^{(b,e,r)} \mathsf{H}\langle\!\langle x \rangle\!\rangle : \tau \quad \Gamma_u \vdash^{(b_u,e_u,r_u)} u : \sigma \quad \sigma \in \mathsf{M}}{x : \mathsf{M} \setminus \sigma; \Gamma \uplus \Gamma_u \vdash^{(b+b_u, e+e_u, r+r_u-1)} \mathsf{H}\langle\!\langle u \rangle\!\rangle : \tau} \ \texttt{partial-subs}$$

Subject reduction and correctness. Quantitative subject reduction is also refined, by taking into account the fact that now there are two evaluation steps, whose numbers are traced by two different indices.

Proposition 6.6 (Quantitative subject reduction for *lhd*). *If $\Phi \triangleright \Gamma \vdash^{(b,e,r)} t{:}\tau$ then*
(1) If $t \to_{\mathsf{m}} u$ then $b \geq 2$ and there is a typing Φ' such that $\Phi' \triangleright \Gamma \vdash^{(b-2,e,r)} u{:}\tau$.
(2) If $t \to_{\mathsf{e}} u$ then $e \geq 1$ and there is a typing Φ' such that $\Phi' \triangleright \Gamma \vdash^{(b,e-1,r)} u{:}\tau$.

The proof is by induction on $t \to_{\mathsf{m}} u$ and $t \to_{\mathsf{e}} u$, using Lemma 6.5. Note that quantitative subject reduction does not assume that the typing derivation is tight: as for the head case, the tight hypothesis is only used for the study of normal forms—it is needed for subject reduction / expansion only if evaluation can take place inside arguments, as in the leftmost and maximal cases.

According to the spirit of tight typings, linear head correctness does not only provide the size of (linear head) normal forms, but also the lengths of evaluation sequences to (linear head) normal form: the two first integers b and e in the final judgement count exactly the total number of evaluation steps to (linear-head) normal form.

Theorem 6.7 (Tight correctness for *lhd*). *Let $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t : \tau$ be a tight derivation. Then there exists p such that $t \to_{lhd}^{b/2+e} p$, $\texttt{normal}_{lhd}(p)$ and $|p|_{lhd} = r$. Moreover, if $\tau = \texttt{neutral}$ then $\texttt{neutral}_{lhd}(p)$.*

6.2 Tight Completeness

As in the case of head and LO evaluation the completeness proof is based on the following properties: typability of linear head normal forms, interaction between (linear head) anti-substitution and typings, and subject expansion. The proofs are analogous to those of the completeness for head and LO evaluation, up to the changes for the linear case, that are instead analogous to those of the correctness of the previous subsection. The statements follow.

Proposition 6.8 (Linear head normal forms are tightly typable for *lhd*). *Let t be such that $\texttt{normal}_{lhd}(t)$. Then there exists a tight typing $\Phi \triangleright_{lhd} \Gamma \vdash^{(0,0,|t|_{lhd})} t : \tau$. Moreover, if $\texttt{neutral}_{lhd}(t)$ then $\tau = \texttt{neutral}$, and if $\texttt{abs}_{lhd}(t)$ then $\tau = \texttt{abs}$.*

Lemma 6.9 (Partial anti-substitution and typings for *lhd*). *Let $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} \mathsf{H}\langle\!\langle u \rangle\!\rangle : \tau$, where $x \notin u$. Then there exists*

- *a type σ*
- *a typing derivation $\Phi_u \triangleright_{lhd} \Gamma_u \vdash^{(b_u,e_u,r_u)} u : \sigma$*
- *a typing derivation $\Phi_{\mathsf{H}\langle\!\langle x \rangle\!\rangle} \triangleright_{lhd} \Gamma' + x{:}[\sigma] \vdash^{(b',e',r')} \mathsf{H}\langle\!\langle x \rangle\!\rangle : \tau$*

such that

- Typing contexts*: $\Gamma = \Gamma' \uplus \Gamma_u$.*
- Indices*: $(b,e,r) = (b' + b_u, e' + e_u, r' + r_u - 1)$.*

Moreover, if Φ is tight then so are Φ_u and $\Phi_{\mathsf{H}\langle\!\langle x \rangle\!\rangle}$.

Proposition 6.10 (Quantitative subject expansion for *lhd*). *If $\Phi' \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t' : \tau$ then*
(1) If $t \to_{\mathsf{m}} t'$ then there is a derivation $\Phi \triangleright_{lhd} \Gamma \vdash^{(b+2,e,r)} t : \tau$.
(2) If $t \to_{\mathsf{e}} t'$ then there is a derivation $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e+1,r)} t : \tau$.

As for linear head correctness, linear head completeness also *refines* the information provided about the lenghts of the evaluation sequences: the number k of evaluation steps to (linear head) normal form is now split into two integers k_1 and k_2 representing, respectively, the multiplicative and exponential steps in such evaluation sequence.

Theorem 6.11 (Tight completeness for lhd). *Let $t \to_{lhd}^{k} p$, where $\mathtt{normal}_{lhd}(p)$. Then there exists a tight type derivation $\Phi \triangleright_{lhd} \Gamma \vdash^{(2k_1, k_2, |p|_{lhd})} t : \tau$, where $k = k_1 + k_2$. Moreover, if $\mathtt{neutral}_{lhd}(p)$, then $\tau = \mathtt{neutral}$, and if $\mathtt{abs}_{lhd}(p)$ then $\tau = \mathtt{abs}$.*

6.3 Relationship Between Head and Linear Head

The head and linear head strategies are specifications at different granularities of the same notion of evaluation. Their type systems are also closely related—in a sense that we now make explicit, they are the same system.

In order to formalise this relationship we define the transformation $\mathcal{L}$ of hd-derivations into (linear, hence the notation) lhd-derivations as: ax in hd is mapped to ax in lhd, $\mathtt{fun}_r$ in hd is mapped to $\mathtt{fun}_r$ in lhd, $\mathtt{fun}_b$ in hd is mapped to $\mathtt{fun}_b$ in lhd, and so on. This transformation preserves the context and the type of all the typing judgements. Of course, if one restricts the lhd system to λ-terms, there is an inverse transformation $\mathcal{N}$ of lhd-derivations into (non-linear, hence the notation) hd-derivations, defined as expected. Together, the two transformation realise an isomorphism.

Proposition 6.12 (Head isomorphism). *Let t be a λ-term without explicit substitutions. Then*

(1) Non-linear to linear*: if $\Phi \triangleright_{hd} \Gamma \vdash^{(b,r)} t : \tau$ then there exists $e \geq 0$ such that $\mathcal{L}(\Phi) \triangleright_{lhd} \Gamma \vdash^{(b,e,r+1)} t : \tau$. Moreover, $\mathcal{N}(\mathcal{L}(\Phi)) = \Phi$.*

(2) Linear to non-linear*: if $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t : \tau$ then $\mathcal{N}(\Phi) \triangleright_{hd} \Gamma \vdash^{(b,r-1)} t : \tau$. Moreover, $\mathcal{L}(\mathcal{N}(\Phi)) = \Phi$.*

The proof is straightforward.

Morally, the same type system measures both head and linear head evaluations. The difference is that to measure head evaluation and head normal forms one forgets the number of axiom typing rules, that coincides exactly with the number of linear substitution steps, plus 1 for the head variable of the linear normal form. In this sense, multi types more naturally measure linear head evaluation. Roughly, a tight multi type derivation for a term is nothing else but a coding of the evaluation in the LSC, including the normal form itself.

On the number of substitution steps. It is natural to wonder how the index e introduced by $\mathcal{L}$ in Proposition 6.12.1 is related to the other indices b and r. This kind of questions has been studied at length in the literature about reasonable cost models. It is known that $e = O(b^2)$ for *any* λ-term, even for untypable ones, see [Accattoli and Dal Lago 2012] for details. The bound is typically reached by the diverging term $\delta\delta$, which is untypable, but also by the following typable term $t_n := (\lambda x_n. \ldots (\lambda x_1.(\lambda x_0.(x_0 x_1 \ldots x_n))x_1)x_2 \ldots x_n)I$. Indeed, t_n evaluates in $2n$ multiplicative steps (one for turning each β-redex into an ES, and one for each time that the identity comes in head position) and $\Omega(n^2)$ exponential steps.

On terms with ES. Relating typings for λ-terms with ES to typings for ordinary λ-terms is a bit trickier—we only sketch the idea. One needs to introduce the *unfolding* operation $(\cdot)\downarrow : \Lambda_{\mathtt{lsc}} \to \Lambda$ on λ-terms with ES, that turns all ES into meta-level substitutions, producing the underlying ordinary λ-term. For instance, $(x[x\backslash y][y\backslash z])\downarrow = z$. As in Proposition 6.12.2, types are preserved:

Lemma 6.13 (Unfolding and lhd derivations). *Let $t \in \Lambda_{\mathtt{lsc}}$. If $\Phi \triangleright_{lhd} \Gamma \vdash^{(b,e,r)} t : \tau$ then there exists $\Phi' \triangleright_{hd} \Gamma \vdash^{(b,r-1)} t\downarrow : \tau$.*

Note that the indices are also preserved. It is possible to also spell out the relationship between Φ and Φ' (as done in [Kesner et al. 2018]), that simply requires a notion of unfolding of typing derivations, and that collapses on the transformation $\mathcal{N}$ in the case of ordinary λ-terms.

7 LEFTMOST EVALUATION AND MINIMAL TYPINGS

This section focuses on the LO system and on the relationship between tight and tight-free—deemed *traditional*—typings. Contributions are manyfold:

(1) *LO normalisation, revisited*: we revisit the characterisation of LO normalising terms as those typable with *shrinking* typings, that is, those where the empty multi-set has no negative occurrences. The insight is that the shrinking and tight constraints are of a very similar nature, showing that our technique is natural rather than ad-hoc. Moreover, our notion of shrinking derivation can also include the tight constants, thus we provide a strict generalisation of the characterisation in the literature.
(2) *Minimality*: we show that tight typings can be seen as a characterisation of minimal traditional shrinking typings. The insight here is that tight typings are simply a device to focalise what traditional types can already observe in a somewhat more technical way.
(3) *Type bound*: we show that for traditional shrinking typings already the type itself—with no need of the typing derivation—provides a bound on the size of the normal form, and this bound is exact if the typing is minimal. The insight is then the inherent inadequacy of multi types as a tool for reliable complexity measures.

This study is done with respect to LO evaluation because among the case studies of the paper it is the most relevant one for reasonable cost models. It may however be easily adapted, *mutatis mutandis*, to the other systems.

Shrinking typings. It is standard to characterise LO normalising terms as those typable with intersection types without negative occurrences of Ω [Krivine 1993], or, those typable with multi types without occurrence of the empty multi-set $[\,]$ [Bucciarelli et al. 2017]. We call this constraint *shrinking*. We need some basic notions. We use the notation T to denote a *(multi)-type*, that is, either a type τ or a multi-set of types $\mathtt{M}$.

Definition 7.1 (Positive and negative occurrences). *Let T be a (multi)-type. The sets of positive and negative occurrences of T in a type/multi-set of types/typing context are defined by mutual induction as follows:*

$$\frac{}{\tau \in \mathtt{Occ}_+(\tau)} \qquad \frac{\exists \sigma \in \mathtt{M} \text{ such that } T \in \mathtt{Occ}_+(\sigma)}{T \in \mathtt{Occ}_+(\mathtt{M})} \qquad \frac{\exists \sigma \in \mathtt{M} \text{ such that } T \in \mathtt{Occ}_-(\sigma)}{T \in \mathtt{Occ}_-(\mathtt{M})}$$

$$\frac{}{\mathtt{M} \in \mathtt{Occ}_+(\mathtt{M})} \qquad \frac{T \in \mathtt{Occ}_-(\mathtt{M}) \text{ or } T \in \mathtt{Occ}_+(\tau)}{T \in \mathtt{Occ}_+(\mathtt{M} \to \tau)} \qquad \frac{T \in \mathtt{Occ}_+(\mathtt{M}) \text{ or } T \in \mathtt{Occ}_-(\tau)}{T \in \mathtt{Occ}_-(\mathtt{M} \to \tau)}$$

$$\frac{T \in \mathtt{Occ}_+(\mathtt{M}) \text{ or } T \in \mathtt{Occ}_+(\Gamma)}{T \in \mathtt{Occ}_+(x : \mathtt{M}, \Gamma)} \qquad \frac{T \in \mathtt{Occ}_-(\mathtt{M}) \text{ or } T \in \mathtt{Occ}_-(\Gamma)}{T \in \mathtt{Occ}_-(x : \mathtt{M}, \Gamma)}$$

Shrinking typings are defined by imposing a condition on the final judgement of the derivation, similarly to tight typings.

Definition 7.2 (Shrinking typing). *Let $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} t{:}\tau$ be a typing derivation.*

- *the typing context Γ is* shrinking *if $[\,] \notin \mathtt{Occ}_-(\Gamma)$;*
- *the final type τ is* shrinking *if $[\,] \notin \mathtt{Occ}_+(\tau)$;*
- *The typing derivation Φ is* shrinking *if both Γ and τ are shrinking.*

Note that

- *Final judgement*: being shrinking depends only on the final judgement of a typing derivation, and that
- *Tight implies shrinking*: a tight typing derivation is always shrinking.

In this section we also have a close look to traditional derivations without tight constants.

Definition 7.3 (Traditional typings). *Let $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} t{:}\tau$ be a typing derivation. Then Φ is* traditional *if no type occurring in Φ is a tight type (and so rules $\mathtt{fun}_r$ and $\mathtt{app}_r^{lo}$ do not occur in Φ).*

One of our results is that the type itself bounds the size of *lo* normal forms, for traditional typings, according to the following notion of type size.

Definition 7.4 (Type size). *The type size $|\cdot|_{ty}$ of types, multi-sets, typing contexts, and derivations is defined as follows:*

$$\begin{array}{rclrcl} |X|_{ty} & := & 0 & |\mathtt{tight}|_{ty} & := & 0 \\ |\mathsf{M}|_{ty} & := & \sum_{\tau \in \mathsf{M}} |\tau|_{ty} & |\mathsf{M} \to \tau|_{ty} & := & |\mathsf{M}|_{ty} + |\tau|_{ty} + 1 \\ |\epsilon|_{ty} & := & 0 & |x : \mathsf{M}; \Gamma|_{ty} & := & |\mathsf{M}|_{ty} + |\Gamma|_{ty} \end{array}$$

$$|\Phi|_{ty} \quad := \quad |\Gamma|_{ty} + |\tau|_{ty} \qquad \text{if } \Phi \triangleright \Gamma \vdash^{(b,r)} t{:}\tau$$

7.1 Shrinking Correctness

Here we show that shrinking typability is preserved by LO evaluation and that the size of shrinking typings decreases along it—hence the name—so that every shrinkingly typable term is LO normalising. For the sake of completeness, we also show that typability is always preserved, but if the typing is not shrinking then its size may not decrease.

Once more, we follow the abstract schema of the other sections (but replacing *tight* with *shrinking* and obtaining bounds that are less tight). The properties of the typings of normal forms and the substitution lemma for the *lo* system have already been proved (Proposition 3.3 and Lemma 3.5). We deal again with normal forms, however, because we now focus on traditional typings, and show that their type bounds the size of the LO normal form. As in the previous sections, neutral terms play a key role, showing that our isolation of the relevance of neutral terms for characterisation via multi types is not specific to tight types.

Proposition 7.5 (Traditional types bounds the size of neutral and normal terms). *Let $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} t{:}\tau$ be a traditional typing. Then:*

(1) If $\mathtt{neutral}_{lo}(t)$ then $|\tau|_{ty} + |t|_{lo} \leq |\Gamma|_{ty}$.

(2) If $\mathtt{normal}_{lo}(t)$ then $|t|_{lo} \leq |\Phi|_{ty}$.

As usual, shrinking correctness is based on a subject reduction property. In turn, subject reduction depends as usual on a spreading property on neutral terms, that is expressed by the following lemma.

Lemma 7.6 (Neutral terms and positive occurrences). *Let t be such that $\mathtt{neutral}_{lo}(t)$ and $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} t : \tau$ be a typing derivation. Then τ is a positive occurrence of Γ.*

It is interesting to note that this lemma subsumes the *tight spreading on neutral terms* property of Lemma 3.4, showing a nice harmony between the shrinking and tight predicates on derivations. If the typing context Γ is tight, indeed, the fact that τ is a positive occurrence of Γ implies that τ is tight.

Proposition 7.7 (Shrinking subject reduction). *Let $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} t{:}\tau$. If $t \to_{lo} p$ then $b \geq 2$ and there exists Φ' such that $\Phi' \triangleright_{lo} \Gamma \vdash^{(b',r)} p{:}\tau$ with $b' \leq b$ (and so $|\Phi'|_{lo} \leq |\Phi|_{lo}$). Moreover, if Φ is shrinking then $b' \leq b - 2$ (and so $|\Phi'|_{lo} \leq |\Phi|_{lo} - 2$).*

Note that a LO diverging term like $x(\delta\delta)$ is typable in system lo by assigning to x the type $[\,] \to X$ and typing $\delta\delta$ with $[\,]$, and that its type is preserved by LO evaluation, by Proposition 7.7. Note however that the resulting judgement is not shrinking—only shrinkingly typable terms are LO normalising, in fact.

Theorem 7.8 (Shrinking correctness). *Let $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} t : \tau$ be a shrinking derivation. Then there exists a $\to_{lo}$ normal form p and $k \leq b/2$ such that*

(1) Steps*: t $\to_{lo}$-evaluates to p in k steps,* i.e. $t \to_{lo}^{k} p$*;*

(2) Result size*: $|p|_{lo} \leq |\Phi|_{lo} - 2k$;*

Moreover, if Φ is traditional then $|t|_{lo} \leq |\Phi|_{ty}$.

Minimality. The minimality of tight typings is hidden in the statement of the shrinking correctness theorem. By combining together its two points, indeed, we obtain that $|p|_{lo} + 2k \leq |\Phi|_{lo}$, that is, the size of every typing derivation bounds both the number of LO steps and the size of the LO normal form. Consequently, tight typings—whose size is exactly the sum of these two quantities—are minimal typings. To complete the picture, one should show that there also exist traditional shrinking typings that are minimal. We come back to this point at the end of the completeness part.

7.2 Shrinking Completeness

The proof of completeness for shrinking typings also follows, *mutatis mutandis*, the usual schema. Normal forms and anti-substitution have already been treated (Proposition 3.8 and Lemma 3.9). Again, however, we repeat the study of (the existence of typings for) LO normal forms focussing now on traditional typings and on the bound provided by types. Their study is yet another instance of *spreading on (LO) neutral terms*, in this case of the size bound provided by types: for neutral terms the size of the typing context Γ allows bounding both the size of the term and the size of its type, which is stronger than what happens for general LO normal terms.

Proposition 7.9 (Neutral and normal terms have minimal traditional shrinking typings).

(1) If $\mathtt{neutral}_{lo}(t)$ then for every type τ there exists a traditional shrinking typing $\Phi \triangleright_{lo} \Gamma \vdash^{(|t|_{lo},0)} t{:}\tau$ such that $|\tau|_{ty} + |t|_{lo} = |\Gamma|_{ty}$.

(2) If $\mathtt{normal}_{lo}(t)$ then there exists a type τ and a traditional shrinking typing $\Phi \triangleright_{lo} \Gamma \vdash^{(|t|_{lo},0)} t{:}\tau$ such that $|t|_{lo} = |\Phi|_{ty}$.

Note that the typings given by the propositions are minimal because they satisfy $|\Phi|_{lo} = |t|_{lo}$ and $|t|_{lo}$ is a lower bound on the size of the typings of t by Proposition 3.3.1—note that it is also the size of a tight typing of t by Proposition 3.8.

The last bit is a subject expansion property. Note in particular that since β-redexes are typed using traditional rules, the expansion preserves traditional typings.

Proposition 7.10 (Shrinking subject expansion). *If $t \to_{lo} p$ and $\Phi \triangleright_{lo} \Gamma \vdash^{(b,r)} p{:}\tau$ then there exists Φ' such that $\Phi' \triangleright_{lo} \Gamma \vdash^{(b',r)} t{:}\tau$ with $b' \geq b$. Moreover, if Φ is shrinking then $b' \geq b+2$, and if Φ is traditional then Φ' is traditional.*

The completeness theorem then follows.

Theorem 7.11 (Shrinking completeness). *Let $t \to_{lo}^{k} p$ with p such that $\mathtt{normal}_{lo}(p)$. Then there exists a traditional shrinking typing $\Phi \triangleright_{lo} \Gamma \vdash^{(b,0)} t{:}\tau$ such that $k \leq b/2$ and $|p|_{lo} = |\Phi|_{ty}$.*

Minimality, again. We have shown that there exist minimal traditional shrinking (MTS for short) typings of normal forms of the same size of the tight typings (Proposition 7.9). It is possible to lift such a correspondence to all typable terms. This refinement is however left to a longer version of the paper, because it requires additional technicalities, needed to strengthen subject expansion.

Let us nonetheless assume, for the sake of the discussion, that tight types characterise minimal typings for all typable terms. What's the difference between the two approaches then? It is easy to describe MTS typings for normal forms: an explicit description can be extracted from the proof of Proposition 7.9, and essentially it interprets the terms *linearly*, that is, typing arguments of applications only once, similarly to the tight app_r^{lo} rule. For non-normal terms, however, MTS typings can be obtained only *indirectly*: first obtaining a MTS typing of the normal form, and then pulling the MTS property back via subject expansion. Tightness instead is a predicate that applies *directly* to derivations of *any* typable term, and is then a direct alternative to MTS typings.

Type bounds. The fact that for traditional typings the type itself provides a bound on the size of the normal form is a very strong property. It is in particular the starting point for de Carvalho's transfer of the study of bounds to the relational semantics of terms [de Carvalho 2007, 2009]—a term is interpreted as the set of its possible types (including the typing context), that is a notion independent of the typing derivations themselves. Because of the size explosion problem, however, such property also shows that the bounds provided by the relational semantics are doomed to be lax and not really informative.

Relational denotational semantics. As we said in the introduction, multi types can be seen as a syntactic presentation of relational denotational semantics, which is the model obtained by interpreting the λ-calculus into the relational model of linear logic [Bucciarelli and Ehrhard 2001; de Carvalho 2007, 2016; Girard 1988], often considered as a canonical model.

The idea is that the interpretation (or semantics) of a term is simply the set of its types, together with their typing 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

$$[\![t]\!]_{\vec{x}} := \{((\mathsf{M}_1, \ldots, \mathsf{M}_n), \tau) \mid \exists\, \Phi \triangleright_{lo} x_1 : \mathsf{M}_1, \ldots, x_n : \mathsf{M}_n \vdash^{(b,r)} t : \tau\}\,.$$

By subject reduction and expansion, the interpretation $[\![t]\!]_{\vec{x}}$ is an invariant of evaluation, and by correctness and completeness it is non-empty if and only if t is LO normalisable. Said differently, multi types provide an adequate denotational model with respect to the chosen notion of evaluation, here the LO one. If the interpretation is restricted to traditional typing derivations (in the sense of Definition 7.3), then it coincides with the one in the relational model in the literature. General derivations still provide a relational model, but a slightly different one, with the two new types `abs` and `neutral`, whose categorical semantics still has to be studied.

8 CONCLUSIONS

Type systems provide guarantees both *internally* and *externally*. Internally, a typing discipline ensures that a program in isolation has a given desired property. Externally, the property is ensured *compositionally*: plugging a typed program in a typed environment preserves the desired property. Multi types (a.k.a. non-idempotent intersection types) are used in the literature to quantify the resources that are needed to produce normal forms. Minimal typing derivations provide exact upper bounds on the number of β-steps plus the size of the normal form—this is the internal guarantee. Unfortunately, such minimal typings provide almost no compositionality, as they essentially force the program to interact with a linear environment. Non-minimal typings allow compositions with less trivial environments, at the price of laxer bounds.

In this paper we have engineered typing so that, via the use of *tight constants* among base types, some typing judgements express compositional properties of programs while other typing judgements, namely the *tight* ones, provide exact and separate bounds on the lengths of evaluation sequences on the one hand, and on the sizes of normal forms on the other hand. The distinction between the two counts is motivated by the size explosion problem, where the size of terms can grow exponentially with respect to the number of evaluation steps.

We conducted this study, building on some of the ideas in [Bernadet and Graham-Lengrand 2013a], by presenting a flexible and parametric typing framework, which we systematically applied to three evaluation strategies of the pure λ-calculus: head, leftmost-outermost, and maximal.

In the case of leftmost-outermost evaluation, we have also developed the traditional shrinking approach which does not make use of tight constants. One of the results is that the number of (leftmost) evaluation steps can be measured using only the (sizes) of the types of the final typing judgement, in contrast to the size of the *whole* typing derivation. Another point, is the connection between tight typings and minimal shrinking typings without tight constants.

In the case of maximal evaluation, we have circumvented the traditional techniques to show strong normalisation: by focusing on the maximal deterministic strategy, we do not require any use of memory operator or subtyping for abstractions to recover subject reduction.

We have also extended our (pure) typing framework to linear head evaluation, presented in the linear substitution calculus (LSC). The result is that tight typings naturally encode evaluation in the LSC, which can be seen as the natural computing devide behind multi types. In particular, and surprisingly, exact bounds for head and linear head evaluation rely on the same type system.

Different future directions are suggested by our contribution. It is natural to extend this framework to other evaluation strategies such as call-by-value and call-by-need. Richer programming features such as pattern matching, or control operators deserve also special attention.

ACKNOWLEDGMENTS

We are grateful to Alexis Bernadet, Giulio Guerrieri and anonymous reviewers for useful discussions and comments. This work has been partially funded by the ANR JCJC grant COCA HOLA (ANR-16-CE40-004-01).

REFERENCES

Beniamino Accattoli. 2012. An Abstract Factorization Theorem for Explicit Substitutions. In *RTA'12 (LIPIcs)*, Ashish Tiwari (Ed.), Vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 6–21.

Beniamino Accattoli. 2018. (In)Efficiency and Reasonable Cost Models. Invited paper at LSFA 2017, to appear. (2018).

Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. 2014. A nonstandard standardization theorem. In *POPL 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 659–670.

Beniamino Accattoli and Ugo Dal Lago. 2012. On the Invariance of the Unitary Cost Model for Head Reduction. In *RTA'12 (LIPIcs)*, Ashish Tiwari (Ed.), Vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 22–37.

Beniamino Accattoli and Ugo Dal Lago. 2016. (Leftmost-Outermost) Beta-Reduction is Invariant, Indeed. *Logical Methods in Computer Science* 12, 1 (2016).

Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2018. Tight Typings and Split Bounds (Long Version). (2018). http://arxiv.org/abs/1807.02358

Martin Avanzini and Ugo Dal Lago. 2017. Automating sized-type inference for complexity analysis. *PACMPL* 1, ICFP (2017), 43:1–43:29.

Erika De Benedetti and Simona Ronchi Della Rocca. 2016. A type assignment for λ-calculus complete both for FPTIME and strong normalization. *Information and Computation* 248 (2016), 195–214.

Alexis Bernadet and Stéphane Graham-Lengrand. 2013a. A big-step operational semantics via non-idempotent intersection types. (2013). https://hal.archives-ouvertes.fr/hal-01830232 Technical Report.

Alexis Bernadet and Stéphane Graham-Lengrand. 2013b. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science* 9, 4 (2013).

Alexis Bernadet and Stéphane Lengrand. 2011. Complexity of Strongly Normalising Lambda-Terms via Non-idempotent Intersection Types. In *FoSSaCS'11 (Lecture Notes in Computer Science)*, Martin Hofmann (Ed.), Vol. 6604. Springer, 88–107.

Guy E. Blelloch and John Greiner. 1995. Parallelism in Sequential Functional Languages. In *FPCA'95*, John Williams (Ed.). ACM, 226–237.

Antonio Bucciarelli and Thomas Ehrhard. 2001. On phase semantics and denotational semantics: the exponentials. *Annals of Pure and Applied Logic* 109, 3 (2001), 205–241.

Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. 2012. A relational semantics for parallelism and non-determinism in a functional setting. *Annals of Pure and Applied Logic* 163, 7 (2012), 918–934.

Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. 2014. The Inhabitation Problem for Non-idempotent Intersection Types. In *IFIP-TCS'14 (Lecture Notes in Computer Science)*, Josep Díaz, Ivan Lanese, and Davide Sangiorgi (Eds.), Vol. 8705. Springer, 341–354.

Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. 2017. Non-idempotent intersection types for the Lambda-Calculus. *Logic Journal of the IGPL* 25, 4 (2017), 431–464.

Alberto Carraro and Giulio Guerrieri. 2014. A Semantical and Operational Account of Call-by-Value Solvability. In *FoSSaCS 2014 (Lecture Notes in Computer Science)*, Anca Muscholl (Ed.), Vol. 8412. Springer, 103–118.

Mario Coppo and Mariangiola Dezani-Ciancaglini. 1978. A new type assignment for lambda-terms. *Archive for Mathematical Logic* 19 (1978), 139–156.

Mario Coppo and Mariangiola Dezani-Ciancaglini. 1980. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic* 4 (1980), 685–693.

Vincent Danos and Laurent Regnier. 2004. *Head Linear Reduction.* Technical Report.

Daniel de Carvalho. 2007. *Sémantiques de la logique linéaire et temps de calcul.* Thèse de Doctorat. Université Aix-Marseille II.

Daniel de Carvalho. 2009. Execution Time of lambda-Terms via Denotational Semantics and Intersection Types. *CoRR* abs/0905.4251 (2009).

Daniel de Carvalho. 2016. The Relational Model Is Injective for Multiplicative Exponential Linear Logic. In *CSL 2016, (LIPIcs)*, Jean-Marc Talbot and Laurent Regnier (Eds.), Vol. 62. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 41:1–41:19.

Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. 2011. A semantic measure of the execution time in linear logic. *Theoretical Computer Science* 412, 20 (2011), 1884–1902.

Daniel de Carvalho and Lorenzo Tortora de Falco. 2016. A semantic account of strong normalization in linear logic. *Information and Computation* 248 (2016), 104–129.

Alejandro Díaz-Caro, Giulio Manzonetto, and Michele Pagani. 2013. Call-by-Value Non-determinism in a Linear Logic Type Discipline. In *LFCS 2013 (Lecture Notes in Computer Science)*, Sergei N. Artëmov and Anil Nerode (Eds.), Vol. 7734. Springer, 164–178.

Andrej Dudenhefner and Jakob Rehof. 2017. Intersection type calculi of bounded dimension. In *POPL'17*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 653–665.

Thomas Ehrhard. 2012. Collapsing non-idempotent intersection types. In *CSL'12 (LIPIcs)*, Patrick Cégielski and Arnaud Durand (Eds.), Vol. 16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 259–273.

Thomas Ehrhard and Giulio Guerrieri. 2016. The Bang Calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *PPDP 2016*. ACM, 174–187.

Philippa Gardner. 1994. Discovering Needed Reductions Using Type Theory. In *TACS '94 (Lecture Notes in Computer Science)*, Masami Hagiya and John C. Mitchell (Eds.), Vol. 789. Springer, 555–574.

Jean-Yves Girard. 1988. Normal functors, power series and the λ-calculus. *Annals of Pure and Applied Logic* 37 (1988), 129–177.

Jean-Yves Girard, Paul Taylor, and Yves Lafont. 1989. *Proofs and types.* Cambridge University Press, New York, NY, USA.

Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. 2016. Computing Connected Proof(-Structure)s From Their Taylor Expansion. In *FSCD 2016 (LIPIcs)*, Delia Kesner and Brigitte Pientka (Eds.), Vol. 52. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 20:1–20:18.

Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2012. Resource Aware ML. In *CAV 2012 (Lecture Notes in Computer Science)*, P. Madhusudan and Sanjit A. Seshia (Eds.), Vol. 7358. Springer, 781–786.

Jan Hoffmann and Martin Hofmann. 2010. Amortized Resource Analysis with Polynomial Potential - A Static Inference of Polynomial Bounds for Functional Programs. In *ESOP'10 (Lecture Notes in Computer Science)*, Andrew D. Gordon (Ed.), Vol. 6012. Springer, 287–306.

Martin Hofmann and Steffen Jost. 2003. Static prediction of heap space usage for first-order functional programs. In *POPL 2003*, Alex Aiken and Greg Morrisett (Eds.). ACM, 185–197.

John Hughes, Lars Pareto, and Amr Sabry. 1996. Proving the Correctness of Reactive Systems Using Sized Types. In *POPL'96*, Hans-Juergen Boehm and Guy L. Steele Jr. (Eds.). ACM, 410–423.

Steffen Jost, Pedro B. Vasconcelos, Mário Florido, and Kevin Hammond. 2017. Type-Based Cost Analysis for Lazy Functional Languages. *Journal of Automated Reasoning* 59, 1 (2017), 87–120.

Delia Kesner and Daniel Ventura. 2014. Quantitative Types for the Linear Substitution Calculus. In *IFIP-TCS'14 (Lecture Notes in Computer Science)*, Josep Díaz, Ivan Lanese, and Davide Sangiorgi (Eds.), Vol. 8705. Springer, 296–310.

Delia Kesner and Daniel Ventura. 2015. A Resource Aware Computational Interpretation for Herbelin's Syntax. In *ICTAC 2015 (Lecture Notes in Computer Science)*, Martin Leucker, Camilo Rueda, and Frank D. Valencia (Eds.), Vol. 9399. Springer, 388–403.

Delia Kesner and Pierre Vial. 2017. Types as Resources for Classical Natural Deduction. In *FSCD 2017 (LIPIcs)*, Dale Miller (Ed.), Vol. 84. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 24:1–24:17.

Delia Kesner, Andrés Viso, and Alejandro Ríos. 2018. Call-by-need, neededness and all that. In *FoSSaCS'18 (Lecture Notes in Computer Science)*, Christel Baier and Ugo Dal Lago (Eds.), Vol. 10803. Springer, 241–257.

Assaf J. Kfoury. 2000. A linearization of the Lambda-calculus and consequences. *Journal of Logic and Computation* 10, 3 (2000), 411–436.

Jan Willem Klop. 1980. *Combinatory Reduction Systems.* PhD thesis. Utrecht University.

Jean-Louis Krivine. 1993. *Lambda-calculus, types and models.* Ellis Horwood.

Gianfranco Mascari and Marco Pedicini. 1994. Head Linear Reduction and Pure Proof Net Extraction. *Theoretical Computer Science* 135, 1 (1994), 111–137.

Damiano Mazza, Luc Pellissier, and Pierre Vial. 2018. Polyadic approximations, fibrations and intersection types. *PACMPL* 2, POPL (2018), 6:1–6:28.

Robin Milner. 2007. Local Bigraphs and Confluence: Two Conjectures. *Electronic Notes in Theoretical Computer Science* 175, 3 (2007), 65–73.

Peter Møller Neergaard and Harry G. Mairson. 2004. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In *ICFP 2004*, Chris Okasaki and Kathleen Fisher (Eds.). ACM, 138–149.

C.-H. Luke Ong. 2017. Quantitative semantics of the lambda calculus: Some generalisations of the relational model. In *LICS 2017*, Joel Ouaknine (Ed.). IEEE Computer Society, 1–12.

Luca Paolini, Mauro Piccolo, and Simona Ronchi Della Rocca. 2017. Essential and relational models. *Mathematical Structures in Computer Science* 27, 5 (2017), 626–650.

Álvaro J. Rebón Portillo, Kevin Hammond, Hans-Wolfgang Loidl, and Pedro B. Vasconcelos. 2002. Cost Analysis Using Automatic Size and Time Inference. In *IFL 2002 (Lecture Notes in Computer Science)*, Ricardo Pena and Thomas Arts (Eds.), Vol. 2670. Springer, 232–248.

Garrel Pottinger. 1980. A Type Assignment for The Strongly Normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J.P. Seldin and J.R. Hindley (Eds.). Academic Press, 561–578.

Hugo R. Simões, Kevin Hammond, Mário Florido, and Pedro B. Vasconcelos. 2007. Using Intersection Types for Cost-Analysis of Higher-Order Polymorphic Functional Programs. In *TYPES 2006, Revised Selected Papers (Lecture Notes in Computer Science)*, Thorsten Altenkirch and Conor McBride (Eds.), Vol. 4502. Springer, 221–236.

Pawel Urzyczyn. 1999. The Emptiness Problem for Intersection Types. *Journal of Symbolic Logic* 64, 3 (1999), 1195–1215.

Pawel Urzyczyn. 2009. Inhabitation of Low-Rank Intersection Types. In *TLCA 2009 (Lecture Notes in Computer Science)*, Pierre-Louis Curien (Ed.), Vol. 5608. Springer, 356–370.

Femke van Raamsdonk, Paula Severi, Morten Heine Sørensen, and Hongwei Xi. 1999. Perpetual Reductions in Lambda-Calculus. *Information and Computation* 149, 2 (1999), 173–225.

Pedro B. Vasconcelos and Kevin Hammond. 2004. Inferring Cost Equations for Recursive, Polymorphic and Higher-Order Functional Programs. In *IFL 2003, Revised Papers (Lecture Notes in Computer Science)*, Vol. 3145. Springer, 86–101.