Multi Types and Reasonable Space

BENIAMINO ACCATTOLI, Inria & LIX, École Polytechnique, UMR 7161, France
UGO DAL LAGO, Università di Bologna, Italy and Inria Sophia Antipolis, France
GABRIELE VANONI, Università di Bologna, Italy and Inria Sophia Antipolis, France

Accattoli, Dal Lago, and Vanoni have recently proved that the space used by the *Space KAM*, a variant of the Krivine abstract machine, is a reasonable space cost model for the λ-calculus accounting for logarithmic space, solving a longstanding open problem. In this paper, we provide a new system of *multi types* (a variant of intersection types) and extract from multi type derivations the space used by the Space KAM, capturing into a type system the space complexity of the abstract machine. Additionally, we show how to capture also the *time* of the Space KAM, which is a reasonable time cost model, via minor changes to the type system.

CCS Concepts: • **Theory of computation** → **Lambda calculus**; **Abstract machines**; **Type theory**.

Additional Key Words and Phrases: lambda calculus, intersection types, cost models, abstract machines, space complexity

ACM Reference Format:
Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2022. Multi Types and Reasonable Space. *Proc. ACM Program. Lang.* 6, ICFP, Article 119 (August 2022), 27 pages. https://doi.org/10.1145/3547650

1 INTRODUCTION

Type systems are a key aspect of programming languages, ensuring good behavior during the execution of programs, such as absence of errors, termination, or properties such as productivity, safety, and reachability properties. For functional programming languages, types tend to have the structure of a logical system, to the point that, in the paradigmatic case of the simply typed λ-calculus, the type system is *isomorphic* to the natural deduction presentation of intuitionistic logic. This is the celebrated *Curry-Howard correspondence*.

In the theory of the λ-calculus, there is another use of types that has been studied at length by experts, but it is less known to the wider research community on programming languages. Historically, it was introduced by Coppo and Dezani-Ciancaglini as the study of *intersection types* [Coppo and Dezani-Ciancaglini 1978]. At first sight, such types are simply a slight variation on simple types, in which one can assign a finite set of types to a term, thus providing a form of finite polymorphism. While some forms of intersection types can be used for programming purposes (see Dezani-Ciancaglini et al. [2018]; Freeman and Pfenning [1991]; Frisch et al. [2008]; Pierce [1997] or the recent survey by Bono and Dezani-Ciancaglini [2020]), they have mainly played a more theoretical role, so far.

Similarly to simple types, intersection types ensure termination. In contrast to most notions of types, however, they also *characterize* termination, that is, they type *all* terminating λ-terms.

Authors' addresses: Beniamino Accattoli, LIX, Inria & LIX, École Polytechnique, UMR 7161, France, beniamino.accattoli@inria.fr; Ugo Dal Lago, Università di Bologna, Italy and Inria Sophia Antipolis, France, ugo.dallago@unibo.it; Gabriele Vanoni, Università di Bologna, Italy and Inria Sophia Antipolis, France, gabriele.vanoni2@unibo.it.

2475-1421/2022/8-ART119
https://doi.org/10.1145/3547650

Intersection types have proved to be remarkably flexible, as by tuning details of the type system it is possible to characterize different forms of termination (weak/strong full normalization, head/weak/call-by-value evaluation) for various kinds of λ-calculi (with pattern matching, state, probabilities, and more). Termination being only *semi-decidable*, type inference cannot be decidable, which is why standard intersection types are somewhat incompatible with programming practice. As already pointed out, restricted forms of intersection types do find applications in programming languages, but the restrictions usually break the characterization of termination, namely their most peculiar trait.

Abstracting Computation. The impractical aspect of intersection types can be understood as coming from the fact that inferring a type for t is very similar to normalizing t. There is a variant of intersection types where the idempotency of intersections $A \cap A = A$ is dropped, which we refer to as *multi types*, because non-idempotent intersections can be seen as multi-sets. Neergaard and Mairson [2004] show that, for multi types, type inference is *equivalent* to normalization of the term to type. Careful here: this is *not* a Curry-Howard correspondence, where normalization corresponds to cut (or detour) elimination from the type derivation, which goes through *many* derivations. Here normalization corresponds to inference of a *single* type derivation, without any rewriting involved.

The undecidability of type inference is balanced by the fact that, intuitively, multi types fully capture the computing mechanism itself, despite looking very different from a computing device. The λ-calculus is already a quite abstract notion of computational model, if compared with Turing or random access machines, but it retains the idea of computation as a state transition system. Multi types push things a level further, as the entire dynamics of normalization is captured by a single *static* type derivation.

Bounding Computation. The literature has not really focussed on the relationship between the two processes of normalization and multi type inference, because such a type inference is impractical. Instead, multi type derivations have been used as *exhaustive transcriptions* of the normalization process, from which to extract information about the normalization process itself. The information usually amounts to bound

(1) the number of steps to normal form, and
(2) the size of the normal form itself,

according to various notions of reduction (weak head/head/leftmost, for instance). The first such results are de Carvalho's bounds on the length of runs of the Krivine abstract machine [Krivine 2007] (shortened to KAM) and of an extension of the KAM to (strong) leftmost evaluation [de Carvalho 2007, 2018]. Such a *de Carvalho correspondence* was shortly afterwards extended to strong normalization by Bernadet and Graham-Lengrand [2013] and to linear logic proof nets by de Carvalho et al. [2011]. The interest in such a correspondence is that it provides an abstract viewpoint on quantitative operational aspects. Moreover, the set of multi type judgment for a term t is a syntactic presentation of the relational semantics of t, a paradigmatic denotational model of the λ-calculus.

Multi Types and Reasonable Cost Models. When de Carvalho developed his correspondence in 2007, it was known that the number of steps in the weak λ-calculus is a reasonable time cost model (where *reasonable* roughly means equivalent up to a *polynomial* to the time cost model of Turing machines), as first proved by Blelloch and Greiner [1995]. The reasonable time cost model of the strong (that is, unrestricted) λ-calculus was instead unclear (because the techniques for the weak case do not work in the strong case), thus it was also unclear to what extent the bounds extracted from multi type derivations are related to the time complexity of λ-terms. In

2014, Accattoli and Dal Lago showed that the number of leftmost β steps is a reasonable time cost model for strong evaluation [Accattoli and Dal Lago 2016], clarifying the situation. In 2018, Accattoli, Graham-Lengrand, and Kesner revisited the de Carvalho correspondence and refined it according to the new understanding of time for the λ-calculus [Accattoli et al. 2018]. Their work triggered a systematic extension of the de Carvalho correspondence to many λ-calculi and abstract machines [Accattoli et al. 2021a; Accattoli and Guerrieri 2018; Accattoli et al. 2019; Alves et al. 2019; Bucciarelli et al. 2020; Dal Lago et al. 2021; Kesner et al. 2021; Kesner and Vial 2020; Kesner and Viso 2022].

In a recent work, Accattoli et al. [2021b] pushed the correspondence into a new direction, by extracting the first *space* bounds from multi type derivations. Additionally, they use these bounds to disprove a conjecture about reasonable space for the λ-calculus. Namely, their bounds apply to Mackie's and Danos & Regnier's *interaction abstract machine* [Accattoli et al. 2020a; Danos et al. 1996; Danos and Regnier 1999; Mackie 1995] (which is the computational machinery behind Girard's Geometry of Interaction [Girard 1989] and Abramsky, Jagadeesan, and Malacaria's game semantics [Abramsky et al. 2000]) and prove that its use of space—that was conjectured to be reasonable—is in fact *unreasonable.*

Reasonable Space. It is only very recently that results about reasonable space for the λ-calculus have started to appear, under the impulse of the advances about reasonable time. Forster, Kunze, and Roth showed that the maximum size of λ-terms during evaluation—the natural space cost model—is reasonable [Forster et al. 2020] (that is, it is *linearly* related to the space of Turing machines, as prescribed by Slot and van Emde Boas [Slot and van Emde Boas 1988; van Emde Boas 1990]). Such a notion of space, however, cannot account for logarithmic space, which is the smallest robust space class, playing the role of P for space. To study logarithmic space, indeed, one needs a separation between *input* space and *work* space, which cannot be modeled taking as space the size of λ-terms. Very recently, Accattoli, Dal Lago, and Vanoni have also shown that the space used by the *Space KAM*, a variant over the KAM, is a reasonable space cost model accounting for logarithmic space [Accattoli et al. 2022a], solving the longstanding open problem. The Space KAM tweaks the KAM in two orthogonal ways:

(1) *Space optimizations*: it performs two space optimizations, *environment unchaining* and *eager garbage collection.* Unchaining is a folklore optimization that prevents space leaks, that is, it keeps the environment compact. Eager garbage collection maximizes the reuse of space. Essentially, alive closures are compacted and dead closures are removed as soon as possible.
(2) *No environment sharing nor pointers*: the sharing of environments used in the implementation of the KAM is *disabled*, and additionally environments (as well as the argument stack) are *not* represented as linked lists (via pointers). Environments and stacks are instead simply unstructured *strings.* These changes are somewhat counter-intuitive but they are in fact mandatory for reasonable space, see [Accattoli et al. 2022a] for extensive discussions.

Accattoli, Dal Lago, and Vanoni also show results about two *time* cost models for the Space KAM: if time is defined as the number of β steps (which is the natural, high-level cost model), then the machine is unreasonable (more precisely, the overhead of the machine is not polynomial), while if time is considered as the time taken by the Space KAM implementation, then it turns out to be a reasonable cost model. Therefore, the Space KAM is reasonable for both time and space, even if for such a simultaneous reasonability one has to pick an unusual and low-level notion of time.

Multi Types and Reasonable Space. In this paper, we develop a de Carvalho correspondence for the Space KAM from which we extract the *space* cost of machine runs. It is the first such correspondence measuring a reasonable notion of space for the λ-calculus. And as it is typical of intersection and multi types, the type system characterizes—and thus provides the *exact* space consumption—for *all* terminating λ-terms (with respect to (weak) CbN evaluation).

De Carvalho seminal work is about the KAM, and relies on the fact that one can see the KAM transition rules as deductive rules of the multi type system. Since the Space KAM is an optimized machine, with modified transitions, we do modify the type system in order to obtain the same correspondence. The key points of the type system are the following ones:

- *Unusual weakening*: the handling of garbage collection induces an unusual approach to *weakening*. The reason is that the Space KAM and the type system implement *weak* evaluation, which does not inspect abstraction bodies, but the definition of garbage is *strong*, as it depends also on occurrences inside abstractions.
- *Indices*: multi types associated to arguments have an index, which is simply a natural number. These indices reflect the size of the *closure* that shall be built for that argument by the Space KAM. They seem arbitrary at first, but if a closed term is typable with a ground type then the indices are uniquely determined, and they capture *exactly* the size of closures. Judgments carry a *weight*, which is another natural number, this time corresponding to the maximum space used by the Space KAM in the run represented by the derivation ending on that judgment.
- *Soundness proof*: soundness of the type system—that is, *t typable with weight w implies that the Space KAM run on t ends and has space cost w*—follows a slightly unusual pattern. We merge together the statements of subject reduction and soundness, which are usually disentangled in the literature. The merge helps proving the space bound.
- *Abstraction of the space cost*: the type system actually counts *the maximum number of closures* used by the Space KAM. Roughly, the *actual space* is obtained by multiplying such a number by the logarithm of the size of the initial term, which corresponds to the size of the *code pointer* in every closure. More precisely, the simulation of Turing machines within a linear overhead in [Accattoli et al. 2022a] rests on some technicalities for which not all code pointers have the same size. Here, we abstract away from those technicalities, and simply count the number of closures. Still, by fine tuning the type system, at the cost of losing clearness, one could recover all the information.

A further contribution of the paper is that, after the study of space, we slightly modify the weights on judgments as to extract the exact *time* cost of runs, instead of the space cost. Therefore, we are able to measure both reasonable time and reasonable space using the same underlying type system decorated with different weights.

Abstracting the Space KAM. For proving that the two type systems capture the time and the space of the Space KAM, we extend the typing rules the states and data structures of the Space KAM. The exact bounds for a term t, however, can be read out *directly* from the typing of t, with no need to type states of the Space KAM. Thus, the extension of the type system to machine states is only a technical tool for the proof of the correspondence. This fact shows that the type system allows us to manipulate the time and space complexities of a term abstracting away from the machine on which these measures are defined. This is the main contribution of the paper.

2 PRELIMINARIES

Here we gently introduce the technical tools that are going to be used and refined along the paper, namely the λ-calculus, Krivine's abstract machine, and multi types.

Reasonable Cost Models. Before starting with the technical discussion, we recall Slot and van Emde Boas' invariance thesis [Slot and van Emde Boas 1988] (also called strong, extended, efficient, modern, or complexity-theoretic Church(-Turing) thesis in the literature), that states: *reasonable computational theories*[1] *simulate each other with polynomially bounded overhead in time and a linear overhead in space.* The rationale behind this thesis is that, under these constraints, complexity classes such as LOGSPACE, P, and PSPACE become robust, i.e. theory-independent. We invite the reader to consult [Accattoli 2017; Accattoli et al. 2022a] for more details.

2.1 The Closed Call-by-Name λ-Calculus

Let $\mathcal{V}$ be a countable set of variables. Terms of the *λ-calculus* Λ are defined as follows:

$$\lambda\text{-TERMS} \qquad t,u,r \quad ::= \quad x \in \mathcal{V} \;\mid\; \lambda x.t \;\mid\; tu.$$

Free and *bound variables* are defined as usual: $\lambda x.t$ binds x in t. Terms are considered modulo α-equivalence, and $t\{x{\leftarrow}u\}$ denotes capture-avoiding (meta-level) substitution of all the free occurrences of x for u in t. The operational semantics of the λ-calculus is given by just one rewriting rule, called the β-rule:

$$(\lambda x.t)u \mapsto_{\beta} t\{x{\leftarrow}u\}$$

Terms like $(\lambda x.t)u$, called reducible expressions, or *redexes*, can occur as sub-terms inside a bigger term. Restricting the β-rule to only some of the redexes, i.e. defining a *reduction strategy*, gives rise to the different λ-calculi. In this paper we deal mainly with a very simple fragment, which we call Closed Call-by-Name (shortened to Closed CbN). It is dubbed *closed*, because we consider only closed terms, i.e. terms which do not have any free variable, and *call-by-name*, because arguments are substituted inside function bodies unevaluated. Moreover, Closed CbN is a *weak* strategy, sometimes called *weak head reduction*, because it does not evaluate under abstractions[2] (which are its all and only normal forms), and because it evaluates only the head redex. We can define Closed CbN reduction via the following rule that can only be applied at top level:

$$(\lambda x.t)ur_1 \dots r_h \quad \rightarrow_{wh} \quad t\{x{\leftarrow}u\}r_1 \dots r_h \quad h \geq 0.$$

For instance, we have $(\lambda x.yxx)t \rightarrow_{wh} \lambda y.tt$ and $(\lambda x.yxx)tu \rightarrow_{wh} (\lambda y.tt)u$ but $r((\lambda x.yxx)tu) \not\rightarrow_{wh} r((\lambda y.tt)u)$ and $\lambda z.((\lambda x.yxx)tu) \not\rightarrow_{wh} \lambda z.((\lambda y.tt)u)$.

It is easily seen that weak head reduction is deterministic.

Example 2.1. In the remainder of this paper, we are going to present different evaluation mechanisms for the λ-calculus. For each of them, we are going to develop a complete example. Each of these examples is based on the execution of the λ-term $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$ where $\mathsf{I} := \lambda a.a$ is the identity combinator. Here we show its Closed CbN evaluation, that takes 3 β-steps.

$$(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I} \rightarrow_{wh} (\lambda y.(\lambda z.\mathsf{I})(\mathsf{I}y))\mathsf{I} \rightarrow_{wh} (\lambda z.\mathsf{I})(\mathsf{I}\mathsf{I}) \rightarrow_{wh} \mathsf{I}$$

2.2 The Krivine Abstract Machine

The KAM is a standard environment-based machine for Closed CbN, often defined as in Fig. 1. The machine evaluates closed λ-terms to weak head normal form via three transitions, the union of which is noted $\rightarrow_{\mathsf{KAM}}$:

- $\rightarrow_{\mathsf{sea}}$ looks for redexes descending on the left of topmost applications of the active term, accumulating arguments on the stack;

[1]Slot and van Emde Boas speak of *machines* rather than *computational theories* because they are mainly discussing Turing and random access machines. Since the λ-calculus is not formulated as a machine, we prefer to speak of *computational theory* (and avoid *computational model* for the collision with *cost model*).

[2]Which is common practice in functional programming languages.

Closures	Environments	Stacks	States
$c ::= (t, e)$	$e ::= \epsilon \mid [x{\leftarrow}c] \cdot e$	$\pi ::= \epsilon \mid c \cdot \pi$	$q ::= (t, e, \pi)$

Term	Env	Stack		Term	Env	Stack	
tu	e	π	$\to_{\mathsf{sea}}$	t	e	$(u, e) \cdot \pi$	
$\lambda x.t$	e	$c \cdot \pi$	$\to_{\beta}$	t	$[x{\leftarrow}c] \cdot e$	π	
x	e	π	$\to_{\mathsf{sub}}$	u	e'	π	if $e(x) = (u, e')$

Fig. 1. KAM data structures and transitions.

- $\to_{\beta}$ fires a β redex (given by an abstraction as active term having as argument the first entry of the stack) but delays the associated meta-level substitutions, adding a corresponding explicit substitution to the environment;
- $\to_{\mathsf{sub}}$ is a form of micro-step substitution: when the active term is x, the machine looks up the environment and retrieves the delayed replacement for x.

The data structures used by the KAM are *local environments*, *closures*, and a *stack*. *Local environments*, that we shall simply refer to as *environments*, are defined by mutual induction with *closures*. The idea is that every (potentially open) term t occurring in a state comes with an environment e that *closes* it, thus forming a closure $c = (t, e)$, and, in turn, environments are lists of entries $[x{\leftarrow}c]$ associating to each open variable x of t, a closure c i.e., intuitively, a closed term. The *stack* simply collects the closures associated to the arguments met during the search for β-redexes.

A state q of the KAM is the pair (c, π) of a closure c and a stack π, but we rather see it as a triple (t, e, π) by spelling out the two components of the closure $c = (t, e)$. Initial states of the KAM are defined as $\mathsf{init}(t_0) := (t_0, \epsilon, \epsilon)$ (where t_0 is a closed λ-term, and also the code). A state is *final* if no transitions apply. A *run* $\rho : q \to^* q'$ is a possibly empty sequence of transitions, the length of which is noted $|\rho|$. If a is a transition label, $|\rho|_a$ is the number of a transitions in ρ. An *initial run* is a run from an initial state $\mathsf{init}(t)$, and it is also called *a run from t*. A state q is *reachable* if it is the target state of an initial run. A *complete run* is an initial run ending on a final state. In a state (t, e, π) that is reachable by an initial state $\mathsf{init}(t_0)$, t is always a sub-term of t_0. For this reason t_0 has a special role during the execution. Indeed, any sub-term t of t_0 can be seen as a pointer to t_0 which we call the *initial immutable code*.

The key point in the proof of the correctness of the KAM is that there is a bijection between $\to_{wh}$ steps and $\to_{\beta}$ transitions, so that we can identify the two. In particular, KAM states can be *decoded* to λ-terms via the following decoding function $\cdot{\downarrow}$ for closures and states:

$$\begin{array}{rrcl@{\qquad}rcl}
\textsc{Closures} & (t,\epsilon){\downarrow} & := & t & (t,[x{\leftarrow}c]{\cdot}e){\downarrow} & := & (t\{x{\leftarrow}c{\downarrow}\},e){\downarrow} \\
\textsc{States} & (t,e,\epsilon){\downarrow} & := & (t,e){\downarrow} & (t,e,\pi{\cdot}c){\downarrow} & := & (t,e,\pi){\downarrow}\,c{\downarrow}
\end{array}$$

Theorem 2.2 (KAM Correctness [Accattoli et al. 2014]). *The KAM implements Closed CbN, that is, there is a complete $\to_{wh}$-sequence $t \to_{wh}^{n} u$ if and only if there is a complete run $\rho : \mathsf{init}(t) {\to_{\mathrm{KAM}}}^* q$ such that $q{\downarrow} = u$ and $|\rho|_{\beta} = n$.*

Example 2.3. We provide the KAM execution of our example term $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$.

Term	Environment	Stack	
$(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$	ϵ	ϵ	$\to_{\mathsf{sea}}$
$\lambda x.(\lambda y.(\lambda z.x)(xy))x$	ϵ	(I, ϵ)	$\to_\beta$
$(\lambda y.(\lambda z.x)(xy))x$	$[x{\leftarrow}(\mathsf{I}, \epsilon)]$	ϵ	$\to_{\mathsf{sea}}$
$\lambda y.(\lambda z.x)(xy)$	$[x{\leftarrow}(\mathsf{I}, \epsilon)]$	$(x, [x{\leftarrow}(\mathsf{I}, \epsilon)])$	$\to_\beta$
$(\lambda z.x)(xy)$	$[y{\leftarrow}(x, [x{\leftarrow}(\mathsf{I}, \epsilon)])] \cdot [x{\leftarrow}(\mathsf{I}, \epsilon)] =: e$	ϵ	$\to_{\mathsf{sea}}$
$\lambda z.x$	$[y{\leftarrow}(x, [x{\leftarrow}(\mathsf{I}, \epsilon)])] \cdot [x{\leftarrow}(\mathsf{I}, \epsilon)]$	(xy, e)	$\to_\beta$
x	$[z{\leftarrow}(xy, e)] \cdot [y{\leftarrow}(x, [x{\leftarrow}(\mathsf{I}, \epsilon)])] \cdot [x{\leftarrow}(\mathsf{I}, \epsilon)]$	ϵ	$\to_{\mathsf{sub}}$
I	ϵ	ϵ	

We can observe that the evaluation takes 7 steps, of which 3 are β-steps, 3 are $\to_{\mathsf{sea}}$-steps, and 1 is a $\to_{\mathsf{sub}}$-step.

2.3 Closed CbN Multi Types

We give in this section an introduction to multi types, also known as *non-idempotent intersection types*. First, we consider them from the *qualitative* point of view, and then we refine our analysis *quantitatively*. We will review them quite quickly, inviting the reader to consult [Bucciarelli et al. 2017] about the qualitative part, and [Accattoli et al. 2020b] about the quantitative one. Multi types are deeply related to linear logic in how they keep track of the use of resources, and in particular their grammar is reminiscent of the (call-by-name, in our case) translation $(\cdot)^\dagger$ of intuitionistic logic into linear logic $(A \to B)^\dagger = !A^\dagger \multimap B^\dagger$. Semantically, they can be seen as a syntactical presentation of the relational model [Bucciarelli and Ehrhard 2001] of the λ-calculus, induced by the relational semantics of linear logic via the CbN translation $(\cdot)^\dagger$.

The grammar for types is based on *two* layers of types, defined in a mutually recursive way, linear types A and finite multi sets M of linear types, this can also be considered a non-idempotent variant of van Bakel [1992] strict types.

$$\begin{array}{llll} \textsc{Linear types} & A, A' & ::= & \star \mid M \to A \\ \textsc{Multi types} & M, N & ::= & [A_1, \ldots, A_n] \qquad n \geq 0 \end{array}$$

Note that there is a ground type $\star$, which can be thought as the type of normal forms, that in Closed CbN are precisely abstractions. Note also that arrow (linear) types $M \to A$ can have a multiset only on the left. The empty multiset is noted $[\,]$, and the union of two multisets M and N is noted $M \uplus N$.

Type environments, ranged over by Γ, Δ are partial maps from variables to multi types such that only finitely many variables are mapped to non-empty multi types, and we write $\Gamma = x_1 : M_1, \ldots, x_n : M_n$ if $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$. Given two type environments Γ, Δ, the expression $\Gamma \uplus \Delta$ stands for the type environment assigning to every variable x the multiset $\Gamma(x) \uplus \Delta(x)$.

Type judgments have the form $\Gamma \vdash t : A$ where Γ is a type environment. The typing rules are in Fig. 2; type derivations are noted π and we write $\pi \triangleright \Gamma \vdash t : A$ for a type derivation π with the judgment $\Gamma \vdash t : A$ as its conclusion.

Intuitively, a linear type A corresponds to a single use of a term, while the cardinality of a multi type assigned to an argument t stands for the number of times this term t will be *copied* during the reduction. More precisely, the cardinality corresponds to the number of times that t is *used* by the KAM, that is, the number of times that t is substituted on the head variable during the KAM run. Different uses of t during a run are allowed to have different types: this is precisely the reason why we need the multiset operator. As an example, we show the type derivation for the

$$\frac{}{x:[A]\vdash x:A}\;\textsc{T-Var} \qquad \frac{\Gamma, x:M\vdash t:A}{\Gamma\vdash \lambda x.t:M\to A}\;\text{T-}\lambda \qquad \frac{}{\vdash \lambda x.t:\star}\;\text{T-}\lambda_\star$$
$$\frac{\Gamma\vdash t:[A'_1,\dots,A'_n]\to A \quad [\Delta_i\vdash u:A'_i]_{i\in[1,\dots,n]}}{\Gamma \uplus \biguplus_{i\in[1,\dots,n]}\Delta_i\vdash tu:A}\;\text{T-@}$$

Fig. 2. The multi type system for Closed CbN.

term $(\lambda x.xx)(\lambda y.y)$.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{x:[[\star]\to\star]\vdash x:[\star]\to\star}\;\textsc{T-Var} \quad \cfrac{}{x:[\star]\vdash x:\star}\;\textsc{T-Var}}{x:[[\star]\to\star,\star]\vdash xx:\star}\;\text{T-@}}{\vdash \lambda x.xx:[[\star]\to\star,\star]\to\star}\;\text{T-}\lambda \quad \cfrac{\cfrac{}{y:\star\vdash y:\star}\;\textsc{T-Var}}{\vdash\lambda y.y:[\star]\to\star}\;\text{T-}\lambda \quad \cfrac{}{\vdash\lambda y.y:\star}\;\text{T-}\lambda_\star}{\vdash(\lambda x.xx)(\lambda y.y):\star}\;\text{T-@}$$

The reader can notice that the argument $\lambda y.y$ is typed twice, and with different types. It is typed once with $[\star] \to \star$, when it is used as a function, and once with $\star$, when it is used as a value. Moreover, the finite polymorphism of multi types allows us to type the term $\lambda x.xx$, which is not typable with simple types.

Characterization of Termination. Multi types characterize Closed CbN termination, that is, they type all and only those λ-terms that terminate with respect to Closed CbN. Differently from what happens with idempotent intersection types, such a result has an easy proof with multi types. Soundness, that is, the fact that typable terms terminate, is usually the hard part, as it requires to exhibit a termination argument. For idempotent intersection types one needs to use the reducibility method, which is a quite heavy machinery. For multi types, there is a strikingly simpler argument: a quantitative form of subject reduction holds, stating that the size of the type derivation decreases with each weak head β-step, which then provides the termination measure. A similar argument shall be used in Section 5.

Completeness of the type system, that is, every terminating term is typable, is proved as for idempotent intersection types, by showing that all weak head normal forms are typable and via a subject *expansion* property.

Theorem 2.4 (Characterization of weak head normalization). *A closed term t has weak head normal form if and only if there exists a multi type derivation $\pi \triangleright \vdash t : \star$.*

Multi Types and the KAM Time Consumption. Multi types have been successfully applied in quantitative analyses of normalization, starting with de Carvalho [2007, 2018], who used them to give a bound to the length of KAM runs. De Carvalho's technique can be re-phrased and distilled as a decoration of type derivations with *weights*, that is, cost annotations, following the scheme of Fig. 3. This weight system just measures the size of the type derivation, since for each rule R, it adds 1 to the weight of the tree rooted in R. Please note that the weight assignment is blind to types (which is why terms and types are removed from Fig. 3), and thus relies only on the structure, an in particular the size, of the type derivation. De Carvalho's result can be formulated as follows.

Theorem 2.5 (de Carvalho). *Let t be a closed λ-term. Then there exists a complete KAM run ρ from t of length w if and only if there exists a multi type derivation $\pi \triangleright \overset{w}{\vdash} t : \star$.*

The KAM being deterministic, one has that all derivations $\vdash t : \star$ induce the same weights. Moreover, there is a stronger correspondence between the rules of the type system, and the transitions

$$\frac{}{\vdash^{1}}\ \text{T-Var} \qquad \frac{\vdash^{w}}{\vdash^{w+1}}\ \text{T-}\lambda \qquad \frac{}{\vdash^{0}}\ \text{T-}\lambda_\star \qquad \frac{\vdash^{w} \quad [\vdash^{v_i}]_{i\in[1,\dots,n]}}{\vdash^{w+\sum v_i+1}}\ \text{T-@}$$

Fig. 3. The weight assignment measuring the size of multi type derivations/KAM execution time.

of the KAM, in the special case of closed terms typed with $\star$. Every T-@ rule corresponds to a $\rightarrow_{\text{sea}}$ transition, every T-λ rule corresponds to a $\rightarrow_{\beta}$ transition[3], and every T-Var rule corresponds to a $\rightarrow_{\text{sub}}$ transition. The only T-$\lambda_\star$ rule in a type derivation for t corresponds to the final state of the KAM run on t. The correspondence is deep, each state of the KAM run for a λ-term t corresponds to a type judgment occurrence in the type derivation of t, in such a way that the sub-term of the state is exactly the same sub-term typed by the judgment. In other words, the KAM and multi types *compute* in the *same* way (in the special case of closed terms typed with $\star$), they only differ in how such a computation is presented.

Example 2.6. We provide the weighted multi type derivation of our running example term $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x:[\star]\vdash^{1} x:\star}\ \text{T-Var}}{x:[\star]\vdash^{2} \lambda z.x:[\,]\to\star}\ \text{T-}\lambda}{x:[\star]\vdash^{3} (\lambda z.x)(xy):\star}\ \text{T-@}}{x:[\star]\vdash^{4} \lambda y.(\lambda z.x)(xy):[\,]\to\star}\ \text{T-}\lambda}{x:[\star]\vdash^{5} (\lambda y.(\lambda z.x)(xy))x:\star}\ \text{T-@}}{\vdash^{6} \lambda x.(\lambda y.(\lambda z.x)(xy))x:[\star]\to\star}\ \text{T-}\lambda \qquad \dfrac{}{\vdash^{0} \mathsf{I}:\star}\ \text{T-}\lambda_\star}{\vdash^{7} (\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}:\star}\ \text{T-@}$$

We note that there is a precise correspondence between this type derivation and the KAM execution of the same term. Indeed, the final weight is 7, as the number of KAM transitions. Moreover, each occurrence of rule T-λ corresponds to a transition $\rightarrow_{\beta}$, each occurrence of rule T-@ corresponds to a transition $\rightarrow_{\text{sea}}$, and each occurrence of rule T-Var corresponds to a transition $\rightarrow_{\text{sub}}$.

3 THE SPACE KAM

Here, we present an optimization of the KAM, that we call Space KAM, aiming at space *reasonability*. More specifically, two modifications are implemented in order to achieve reasonability, namely *unchaining* and *eager garbage collection.*

Unchaining. It is a folklore optimization for abstract machines bringing speed-ups with respect to both time and space, used e.g. by Sands et al. [2002], Wand [2007], Friedman et al. [2007], and Sestoft [1997]. Its first systematic study is by Accattoli and Coen [2017], with respect to time. This optimization prevents the creation of chains of *renamings* in environments, that is, of delayed substitutions of variables for variables, of which the simplest shape in the KAM is:

$$[x_0{\leftarrow}(x_1,[x_1{\leftarrow}(x_2,[x_2{\leftarrow}\ldots])])]$$

where the links of the chain are generated by β-redexes having a variable as argument. On some families of terms, these chains keep growing and growing, leading to the quadratic dependency of the number of transitions from $|\rho|_{\beta}$, and to the continuous increase of space consumption. An

[3]This way, multi type derivations can be used to count also the number of $\rightarrow_{wh}$-steps a λ-term needs to normalize, which is the natural time cost model for Closed CbN.

Closures	Environments	Stacks	States
$c ::= (t, e)$	$e ::= \epsilon \mid [x{\leftarrow}c] \cdot e$	$\pi ::= \epsilon \mid c \cdot \pi$	$q ::= (t, e, \pi)$

Term	Env	Stack		Term	Env	Stack	
tx	e	π	$\rightarrow_{\mathsf{sea_v}}$	t	$e\vert_t$	$e(x) \cdot \pi$	
tu	e	π	$\rightarrow_{\mathsf{sea_{\neg v}}}$	t	$e\vert_t$	$(u, e\vert_u) \cdot \pi$	if $u \notin \mathcal{V}$
$\lambda x.t$	e	$c \cdot \pi$	$\rightarrow_{\beta_{\mathsf{w}}}$	t	e	π	if $x \notin \mathsf{fv}(t)$
$\lambda x.t$	e	$c \cdot \pi$	$\rightarrow_{\beta_{\neg\mathsf{w}}}$	t	$[x{\leftarrow}c] \cdot e$	π	if $x \in \mathsf{fv}(t)$
x	e	π	$\rightarrow_{\mathsf{sub}}$	u	e'	π	if $e(x) = (u, e')$

where $e|_t$ denotes the restriction of e to the free variables of t.

Fig. 4. Space KAM data structures and transitions.

example of continuosly growing chain of renamings is easily obtained by computing finite prefixes of the KAM execution of the standard looping term Ω.

Eager Garbage Collection. Beside the malicious chains connected to unchaining, the Naive KAM is not parsimonious with space also because there is no garbage collection (shortened to GC). In transition $\rightarrow_{\mathsf{sub}}$, the current environment is discarded, so something is collected, but this is not enough. It is thus natural to modify the machine as to maximize GC and space re-usage, that is, as to perform it *eagerly.*

Space KAM Transitions and Data Structures. The KAM optimized with both eager GC and unchaining (both optimizations are mandatory for space reasonability) is here called Space KAM and it is defined in Fig. 4. The data structures, namely closures and (local) environments, are defined as for the KAM, the changes concern the machine transitions only. Unchaining is realized by transition $\rightarrow_{\mathsf{sea_v}}$, while eager garbage collection is realized mainly by transition $\rightarrow_{\beta_{\mathsf{w}}}$, which discards the argument if the variable of the β redex does not occur. Transitions $\rightarrow_{\mathsf{sea_{\neg v}}}$ and $\rightarrow_{\mathsf{sea_v}}$ also contribute to implement the GC, by restricting the environment to the occurring variables, when the environment is propagated to sub-terms. As a consequence, we obtain the following invariant.

Lemma 3.1 (Environment domain invariant). *Let q be a Space KAM reachable state. Then* $\mathsf{dom}(e) = \mathsf{fv}(t)$ *for every closure (t, e) in q.*

Because of the invariant, which concerns also the closure given by the active term and the local environment of the state, the variable transition $\rightarrow_{\mathsf{sub}}$ simplifies as follows:

Term	Env	Stack		Term	Env	Stack
x	$[x{\leftarrow}(u, e)]$	π	$\rightarrow_{\mathsf{sub}}$	u	e	π

Implementation and Space Consumption. We consider the Space KAM implemented in a rather unusual way, in order to obtain a space reasonable machine. Indeed, we avoid any form of sharing of data structures. Transitions $\rightarrow_{\mathsf{sea_{\neg v}}}$ and $\rightarrow_{\mathsf{sea_v}}$ actually copy the environment itself, and not just a pointer to it. Moreover, all the data structures are implemented as unstructured strings, i.e. not as linked lists or trees (just think about implementing the Space KAM directly on a multi tape Turing machine, rather than on a RAM). In this way, the machine does not rest on pointers for structuring the stack and the environment data structures, and this is crucial, as such pointers would add an unreasonable space overhead. Pointers do not disappear altogether: pointers to the initial immutable code are used to represent the left component of every closure (that is, the term u in (u, e)), which, by the key *sub-term invariant* of the KAM, is always a sub-term of the initial code. The use of these pointers is another crucial ingredient for space reasonability, as it allows to avoid the copy of sub-terms which would make the code grow during execution (that must be avoided to account for logarithmic space).

The avoidance of data structure pointers is also the reasong why garbage collection can be easily implemented eagerly, as it is assumed by the Space KAM. If one instead adopts a more standard form of *shared* environments (which are based on using pointers) then garbage collection becomes trickier, because of *aliasing*, that is, the fact that shared environments can be referenced more than once. Our approach, instead, does not suffer from this difficulty, since *nothing* is shared. Such a choice of course degrades the time performance, but here we are mainly interested in space behavior. All these implementation choices, necessary to obtain the space reasonability of the Space KAM, are discussed at length in Accattoli et al. [2022a].

Space and Number of Closures of the Space KAM. Intuitively, the space occupied by a state q of the Space KAM during a run of initial immutable code t_0 is the number n of closures in q multiplied by the logarithm $\log |t_0|$ of the size of the initial immutable code t_0, accounting for the code in every closure (that is, the left component), which is represented as a pointer to the initial code (because of the sub-term invariant of the KAM mentioned above).

For the space reasonability result in [Accattoli et al. 2022a], however, a trickier notion of space is used, as the initial code is divided in *two* address spaces and so not every pointer takes space $\log |t_0|$ (some use less space). We abstract away from such a technical aspect, because it is needed to study the representation of Turing machines, but on λ-terms that do not belong to the image of the encoding of Turing machines it is irrelevant. Therefore, we use an abstract notion of space, namely *the number of closures in a state*, which roughly is the space of the Space KAM up to $\log |t_0|$.

Definition 3.2 (Abstract space). The abstract space $|q|$ taken by a Space KAM state q with initial immutable code t_0 is defined on the structure of Space KAM states as follows[4].

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\multicolumn{3}{c}{\textsc{Environments}} & \multicolumn{3}{c}{\textsc{Stacks}} \\
|\epsilon| & := & 0 & |\epsilon| & := & 0 \\
|[x{\leftarrow}c]\cdot e| & := & |c| + |e| & |c \cdot \pi| & := & |c| + |\pi| \\
\\
\multicolumn{3}{c}{\textsc{Closures}} & \multicolumn{3}{c}{\textsc{States}} \\
|(t, e)| & := & 1 + |e| & |(t, e, \pi)| & := & |e| + |\pi|
\end{array}$$

The size a Space KAM run then it is obtained by considering the maximum size of the states reached along that run.

Definition 3.3 (Run Abstract Space). Let $\rho : \mathsf{init}(t_0) \to_{\mathrm{SpKAM}}{}^* q$ be a Space KAM run. Then the (abstract) space consumption of the run ρ is defined as follows:

$$|\rho|_{\mathrm{sp}} := \max_{q' \in \rho} |q'|$$

[4]In the clause about states, we could have written $|(t, e, \pi)| := |e| + |\pi| + 1$, counting also the space for the closure (t, e). Since the difference is just about a constant, we can remove that "+1" without loss of generality. The removal of the +1 shall slightly simplify the extraction of the space usage from multi type derivations in Section 5.

Example 3.4. We provide the Space KAM execution of our example term $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$.

Term	Environment	Stack	
$(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$	ϵ	ϵ	$\to_{\mathsf{sea}_{\neg\mathsf{v}}}$
$\lambda x.(\lambda y.(\lambda z.x)(xy))x$	ϵ	(I,ϵ)	$\to_{\beta_{\neg\mathsf{w}}}$
$(\lambda y.(\lambda z.x)(xy))x$	$[x{\leftarrow}(\mathsf{I},\epsilon)]$	ϵ	$\to_{\mathsf{sea}_{\mathsf{v}}}$
$\lambda y.(\lambda z.x)(xy)$	$[x{\leftarrow}(\mathsf{I},\epsilon)]$	(I,ϵ)	$\to_{\beta_{\neg\mathsf{w}}}$
$(\lambda z.x)(xy)$	$[y{\leftarrow}(\mathsf{I},\epsilon)]\cdot[x{\leftarrow}(\mathsf{I},\epsilon)] =: e$	ϵ	$\to_{\mathsf{sea}_{\neg\mathsf{v}}}$
$\lambda z.x$	$[x{\leftarrow}(\mathsf{I},\epsilon)]$	(xy,e)	$\to_{\beta_{\mathsf{w}}}$
x	$[x{\leftarrow}(\mathsf{I},\epsilon)]$	ϵ	$\to_{\mathsf{sub}}$
I	ϵ	ϵ	

Note that now the environment contains only closures bound to the free variables of the current term. Moreover, closures that will be never used, because the abstracted variable does not occur in the body of the abstraction, as simply deleted, such as (xy, e) in the $\to_{\beta_{\mathsf{w}}}$ transition. Finally, no chains that rename variables are present anymore. In the previous execution, on the KAM, there was one such chain, namely $(x, [x{\leftarrow}(\mathsf{I},\epsilon)])$, which now is just (I,ϵ). From the quantitative point of view, note that the maximum space consumed during the execution is 4 pointers: in the third to last state, the closure in the stack contains e, which is itself an environment with two closures, as defined at the previous line.

4 THE CLOSURE TYPE SYSTEM

Here we define our variant of multi types, dubbed *closure types*, that we are going to use to measure the space consumption of (typable) terms. The definition of types is standard, but for the fact that multi-sets come labeled with an index k. The idea is that multi-sets of types are associated to arguments (according to the call-by-name translation of the λ-calculus into linear logic), and arguments give rise to closures (hence the name closure types): the index represent the size $|c|$ of the closure c (i.e. the number of pointers to implement it) that shall be associated to that argument/multi-set.

$$\begin{array}{lrcll} \textsc{Linear Types} & A, A' & ::= & \star \mid M^k \to A & \\ \textsc{Closure Types} & M^k, N^k & ::= & [A_1,\dots,A_n]^k & k>0, n\geq 0 \end{array}$$

Type judgments, type environments, and type derivations are defined exactly as in the traditional multi type case. Please note that however some subtleties become apparent, regarding how labeled multi sets are handled. The empty multi-set is noted $[\,]^k$ and it also comes labeled with k. Note that k has to be strictly positive. The sum $\uplus$ of multi-sets requires the two multi-sets to have the same index, that is, we have that $M^k \uplus N^k := (M \uplus N)^k$, where on the right hand side we treat M and N as ordinary multi-sets, while $M^k \uplus N^h$ is undefined for $h \neq k$.

Space and Weakenings. The study of space requires an unusual approach to weakenings. In a weak evaluation setting, a judgment $\Gamma \vdash t : A$ usually implies only that $\mathsf{dom}(\Gamma) \subseteq \mathsf{fv}(t)$ and not necessarily that $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$, because there can be untyped free variables, that is, free variables occurring under abstraction that are not touched by weak evaluation and thus not typed. Indeed, given an abstraction $\lambda x.t$ with $\mathsf{fv}(\lambda x.t) \neq \emptyset$, usually one can type it with $\star$, deriving a judgment $\vdash \lambda x.t : \star$ with an *empty* type context. Here this shall not be possible, because in the Space KAM the variables in $\mathsf{fv}(\lambda x.t)$ play a role in the space usage, as they forbid to garbage collect some closures. Therefore, we modify the type system as to enforce the property $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$, even if evaluation is weak, and this is done via an unusual use of weakenings. In particular, we distinguish between a variable $x \notin \mathsf{dom}(\Gamma)$ and a variable $\Gamma(x) = [\,]^{k_x}$ that gets assigned an empty multi-set. The intuition

$$\frac{}{x : [A]^k \overset{k+|A|}{\vdash} x : A}\ \text{T-VAR} \qquad \frac{\mathsf{dom}(\Gamma) = \mathsf{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \overset{|\Gamma|}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma, x : M^k \overset{w}{\vdash} t : A}{\Gamma \overset{w}{\vdash} \lambda x.t : M^k \multimap A}\ \text{T-}\lambda_1 \qquad \frac{\Gamma \overset{w}{\vdash} t : A \quad x \notin \mathsf{dom}(\Gamma)}{\Gamma \overset{\max\{w, |\Gamma|+|A|+k\}}{\vdash} \lambda x.t : [\,]^k \multimap A}\ \text{T-}\lambda_2$$

$$\frac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \le i \le n \quad \#_i\Gamma_i}{\uplus_{i=1}^n \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_1..A_n]^{1+|\uplus_{i=1}^n \Gamma_i|}}\ \text{T-MANY} \qquad \frac{\mathsf{dom}(\Gamma) = \mathsf{fv}(t) \quad \Gamma \text{ is dry}}{\Gamma \overset{0}{\vdash} t : [\,]^{1+|\Gamma|}}\ \text{T-NONE}$$

$$\frac{\Gamma \overset{w}{\vdash} t : M^k \multimap A \quad \Delta \overset{v}{\vdash} u : M^k \quad \Gamma\#\Delta}{\Gamma \uplus \Delta \overset{\max\{w,v\}}{\vdash} tu : A}\ \text{T-@}_1 \qquad \frac{\Gamma \overset{w}{\vdash} t : M^k \multimap A \qquad \Gamma\#x : M^k}{\Gamma \uplus x : M^k \overset{w}{\vdash} tx : A}\ \text{T-@}_2$$

Fig. 5. The closure type system.

is that $x \notin \mathsf{dom}(\Gamma)$ means that x does not appear in the term at all, and it would correspond to x being typed with $[\,]^0$, which is however not a valid type because the index k_x must be > 0. Instead, $\Gamma(x) = [\,]^{k_x}$ means that x appears but it is not going to be used, and it shall nonetheless have an associated closure of size k_x. We point out that also Lengrand et al. [2004] have studied intersection types in a calculus with explicit substitutions and a garbage collection rule.

We shall need a notion of type context assigning the empty multi-set (with a positive index) to all the variable in its domain.

Definition 4.1 (Dry type contexts). A type context Γ is *dry* if for every $x \in \mathsf{dom}(\Gamma)$, there exists k_x such that $\Gamma(x) = [\,]^{k_x}$.

Typing Rules. In order to define the typing rules we need two further notions. First, we need a notion of size of types and type contexts.

$$|\star| := 0 \qquad |M^k \multimap A| := k + |A| \qquad |x : M^k, \Gamma| := k + |\Gamma|$$

Note that for a type context one sums only the indices over the multi-sets, ignoring the size of linear types inside the multi-sets themselves.

To define the type system, we need a predicate over type environments to ensure that they are summable, because multi-sets sum is restricted to multi-sets having the same index.

Definition 4.2. Two type environments Γ and Δ are *summable*, noted $\Gamma\#\Delta$, if when $\Gamma(x) = M^k$ and $\Delta(x) = N^h$ then $k = h$, for all $x \in \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta)$. The notion can be naturally generalized to an arbitrary number of type environment Γ_i as $\#_i\Gamma_i$.

The typing rules are in Fig. 5. For now, just ignore the weights. As in the case of the multi type system and of the KAM, we have crafted this system in such a way that given a type derivation π for a term t, there is a one to one correspondence between the transitions used by the Space KAM run on t and the occurrences of the typing rules (excluded T-MANY and T-NONE) of π, plus the fact that T-$\lambda_\star$ is used to type final states. Namely, $\rightarrow_{\mathsf{sea_v}}$ corresponds to T-@$_2$, $\rightarrow_{\mathsf{sea_{\neg v}}}$ to T-@$_1$, $\rightarrow_{\beta_{\neg w}}$ to T-λ_1, $\rightarrow_{\beta_w}$ to T-λ_2, and $\rightarrow_{\mathsf{sub}}$ to T-VAR. The intuition behind the closure types is that there is a correspondence between Space KAM data structures and the type theoretic side. The idea is that every closure type M^k corresponds to a closure c such that $|c| = k$. In particular, given a state $q = (t, e, \pi)$ and a judgment $\Gamma \vdash t : A$, the type environment $\Gamma := x_1 : M^{k_1} \dots x_n : M^{k_n}$ morally corresponds to the environment $e := [x_1 \leftarrow c_1] \dots [x_n \leftarrow c_n]$ and moreover $|c_i| = k_i$ for each $1 \leq i \leq n$. Similarly, there is a correspondence between the stack and the type A, namely $A := M_1^{k_1} \to \cdots \to M_m^{k_m} \to \star$ corresponds to the stack $\pi := \pi_1 \dots \pi_m$ and moreover $|\pi_i| = k_i$ for each $1 \leq i \leq m$. Essentially, this means that we are able to read from the type derivation of a term t the sizes of all the states reached by the Space KAM evaluating t.

Rules T-many and T-none is where multi-sets are introduced on the right. In order to explain the index on the introduced multi-set, let us first consider T-none. The idea is that t shall be paired by the machine with an environment e as to form a closure $c = (t, e)$. By the invariant of the machine (Lemma 3.1), $\mathsf{dom}(e)$ is exactly $\mathsf{fv}(t)$, that is, e contains a closure for each variable x in $\mathsf{fv}(t)$. Now, each such closure shall have size k_x, where k_x is the index given to $[\,]$ by the dry context Γ. Therefore, $1 + |\Gamma|$ shall correspond to $1 + |e|$, which is exactly the size of c.

For T-many, the reasoning is analogous. Let us explain a point about the quantity $|\uplus_{i=1}^{n} \Gamma_i|$ in its conclusion. An invariant (forthcoming Lemma 5.1) shall guarantee that $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$, whenever $\Gamma \vdash t : A$. Then in T-many, the type contexts Γ_i have all the same domain, and by the summable hypothesis, they all give the same index k to the (potentially different) multi-sets $\Gamma_i(x)$ for a same variable x. Thus the various $|\Gamma_j|$ all coincide (for $j \in \{1, \dots, n\}$) and also coincide with the quantity $|\uplus_{i=1}^{n} \Gamma_i|$ (so one could have as well decorated the conclusion of T-many with the index $|\Gamma_1|$ rather than $|\uplus_{i=1}^{n} \Gamma_i|$, as they coincide).

Some of the rules of the type system seem to introduce some *arbitrary* indices in the conclusions. Namely, rules T-Var and T-λ_2 introduce an arbitrary $k > 0$, and rules T-$\lambda_\star$ and T-none refer to a dry Γ, and dryness only requires the indices to be different from 0 but otherwise arbitrary. Two lemmas in the next section (Lemma 5.2 and Lemma 5.3) shall however guarantee that, if the global derivation in which the rules occur types a closed term with $\star$, such arbitrary choices are in fact strongly constrained, to the point that the indices are actually *uniquely* determined.

We highlight the main differences w.r.t. the traditional multi type system.

- *Weakening.* As we have already observed, the approach to weakenings is slightly more liberal than in multi types. It is implicitly part of rules T-$\lambda_\star$, T-λ_2, and T-none. In rules T-$\lambda_\star$ and T-none, a context Γ can be injected in the conclusion, but only if it is dry, and if its domain coincides with the free variables of the typed term. The situation at the level of the Space KAM for the rule T-$\lambda_\star$ corresponds to final states such as $(\lambda x.y, [y \leftarrow t], \epsilon)$. A closure is indeed present, hence the index $k > 0$, although it will never be used, hence its type being the empty multi-set. In rule T-λ_2, which is not present in the multi type system (because transition $\to_{\beta_w}$ is not a transition of the KAM), the variable x that does not occur free in t is intuitively typed with $[\,]^0$ in the premise in the rule. In the conclusion, however, a type $[\,]^0 \to A$ would not be correct, as the index cannot be 0. Then the rule uses an arbitrary index $k > 0$. The intuition is that the Space KAM closure corresponding to $[\,]$, before being eliminated by rule $\to_{\beta_w}$, has size strictly greater than zero.
  Please notice that all this consideration ensure the invariant for which for any judgment $\Gamma \vdash t : A$ or $\Gamma \vdash t : M^k$ one has $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$.
- *Separate multi-sets rules.* In the traditional multi types rules T-none and T-many are incorporated inside rule T-@. This was impossible to do in the closure type system because the rule T-none was necessarily different from the rule T-many, and not just the special case when there are zero premises. This is because of the way we deal with weakenings: weakening is present just in the rule T-none, while there is not in rule T-many.

Rules T-NONE and T-MANY correspond to the creation of closures. This why the index of the created multi-set is $1 + |\Gamma|$. Morally, this corresponds to the closure (t, e), where t is the typed term and e corresponds to Γ. Hence, we have $|(t, e)| = 1 + |e| = 1 + |\Gamma|$.

- *Unchaining.* The type rule T-@$_2$ is the type theoretic equivalent of the Space KAM transition rule $\rightarrow_{\mathtt{sea_v}}$, that implements the unchaining optimization. If one forgets the indices, the rule is just a specialization of the rule T-@$_1$, from which it can be derived (using also an instance of rule T-VAR). The derived rule, however, would not have the right indices (because it corresponds to the KAM way of computing that does not unchain), which is why we need an additional rule.
- *Coherence.* All the rules which sum type environments have to enforce the coherence relation $\#\Gamma_i$ between the type environments Γ_i in the premises.

The Weight System. The intuition behind the weight system is very simple. The weight of the type derivation π ending in $\vdash t : \star$ should correspond to the space consumed by the complete run ρ of the Space KAM starting from t. We know how to read the size of a Space KAM state $q = (t, e, \pi)$ belonging to ρ out of the corresponding to type judgment $\Gamma \vdash t : A$ belonging to π: simply $|q| = |\Gamma| + |A|$. Then, the space consumed by ρ is the maximum of the sizes of all the type judgments occurring in π. This is exactly what the weighting system is keeping trace of.

Example 4.3. We provide the closure type derivation of our example term $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\star]^1 \vdash^1 x : \star}\,\text{T-VAR}}{x : [\star]^1 \vdash^4 \lambda z.x : [\,]^3 \to \star}\,\text{T-}\lambda_2 \qquad \dfrac{}{x : [\,]^1, y : [\,]^1 \vdash^0 xy : [\,]^3}\,\text{T-NONE}}{x : [\star]^1, y : [\,]^1 \vdash^4 (\lambda z.x)(xy) : \star}\,\text{T-@}_1}{x : [\star]^1 \vdash^4 \lambda y.(\lambda z.x)(xy) : [\,]^1 \to \star}\,\text{T-}\lambda_1}{x : [\star]^1 \vdash^4 (\lambda y.(\lambda z.x)(xy))x : \star}\,\text{T-@}_2}{\vdash^4 \lambda x.(\lambda y.(\lambda z.x)(xy))x : [\star]^1 \to \star}\,\text{T-}\lambda_1 \qquad \dfrac{\dfrac{}{\vdash^0 \mathsf{I} : \star}\,\text{T-}\lambda_\star}{\vdash^0 \mathsf{I} : [\star]^1}\,\text{T-MANY}}{\vdash^4 (\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I} : \star}\,\text{T-@}_1$$

Also in this case, we can observe the precise correspondence between this type derivation and the execution of the Space KAM. Not only rules and transitions are into a one-to-one correspondence, but also stack and environment entries (with their sizes) can be seen, respectively, in types and in type environments. Of course, as a consequence, the final weight is 4, as the space consumption of the Space KAM execution.

5 WEIGHTS CAPTURE THE SPACE OF THE SPACE KAM

This section is devoted to the proof of our main result, that is, the fact that the wieght of a type judgment (assigning type $\star$ to a closed term t) captures the space used by the (complete run on t of the) Space KAM. The proof is carried out in a rather standard way. First, we prove soundness, that is the fact that typable terms terminate, and moreover that their evaluation respects the weight. Second, we prove completeness, that is the fact that all terminating terms are typable.

5.1 Preliminary Properties

The first property that we prove here is qualitative, that is, it does not concern weights or indices[5]. It is simply the already mentioned fact that the domain of type contexts is exactly the set of free variables of the typed term. This is the type analog of the environment domain invariant of the Space KAM.

[5]When a definition or a statement does not rest on weights, we do not report them, for the sake of readability.

$$\frac{\overset{w_x}{\vdash} e(x) : \Gamma(x) \qquad \forall x \in \mathsf{dom}(e)}{\overset{\max\{w_x \mid x \in \mathsf{dom}(e)\}}{\vdash} e : \Gamma}\ \textsc{T-env} \qquad \frac{\overset{w}{\vdash} e : \Gamma \qquad \Gamma \overset{v}{\vdash} t : M^k}{\overset{\max\{w,v\}}{\vdash} (t,e) : M^k}\ \textsc{T-cl}$$

$$\frac{\Gamma \overset{w}{\vdash} t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \overset{u}{\vdash} e : \Gamma \quad \overset{v_i}{\vdash} c_i : M_i^{k_i}}{\overset{\max_i\{w,v_i,u\}}{\vdash} (t,e,c_1\cdots c_n) : \star}\ \textsc{T-st}$$

Fig. 6. Typing rules for the machine components of the Space KAM.

Lemma 5.1 (Type contexts domain invariant).

(1) *If* $\pi \triangleright \Gamma \vdash t : A$ *then* $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$.
(2) *If* $\pi \triangleright \Gamma \vdash t : M$ *then* $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$.

Proof. By induction on π. □

Then, we have two quantitative properties. Before stating them, we need to introduce the definition of typed Space KAM states. The idea is very simple, and it is due to de Carvalho [2018]. In order to state quantitative properties that involve abstract machines, we type the execution of the abstract machine itself. We have already mentioned that type environments correspond to the Space KAM environments, and that closure types correspond to Space KAM closures. In Fig. 6, the connection is made formal.

The next lemma states that the size of a closure is equal to the size of its type, for every closure, and similarly for environments.

Lemma 5.2 (Types capture the size of closures and environments).

(1) *If* $\pi \triangleright \vdash e : \Gamma$ *then* $|e| = |\Gamma|$.
(2) *If* $\pi \triangleright \vdash c : M^k$ *then* $|c| = k$.

The lemma expresses one of the key properties of the type system, which has an almost *magical* feeling. At first sight, indeed, the indices on multi types are completely arbitrary, as the typing rules do not seem to impose strong constraint. Surprisingly, instead, when one builds the type derivation of a closure, then the indices are uniquely determined, and they capture *exactly* the size of the closure.

Proof. By mutual induction on e and c.

(1) π has the following form:

$$\frac{\vdash e(x) : \Gamma(x)}{\vdash e : \Gamma}\ \textsc{T-env}$$

Let $\Gamma(x) = M_x^{k_x}$. By *i.h.* (point 2), $|e(x)| = k_x$. Then $|e| = \sum_{x \in \mathsf{dom}(e)} |e(x)| =_{i.h.} \sum_{x \in \mathsf{dom}(e)} k_x = |\Gamma|$.

(2) π has the following form:

$$\frac{\vdash e : \Gamma \qquad \Gamma \vdash t : M^k}{\vdash (t,e) : M^k}\ \textsc{T-cl}$$

with $c = (t,e)$. By *i.h.* (point 1) applied to e, we have $|e| = |\Gamma|$. Since the typing of t comes necessarily from a rule T-none or T-many, we have $k = 1 + |\Gamma|$. Then $|c| = 1 + |e| =_{i.h.} 1 + |\Gamma| = k$.

□

The second easy property is the fact that the size of a state is given by the size of the types in the judgment for its code, which give the size of the stack and the environment.

Lemma 5.3 (Judgments capture the size of the corresponding state). *If*

$$\frac{\Gamma \vdash t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \vdash c_i : M_i^{k_i} \quad \vdash e : \Gamma}{\vdash (t, c_1 \cdots c_n, e) : \star} \; \textit{T-st}$$

then $|\Gamma| + \sum_{i=1}^{n} k_i = |q|$.

Proof. We have:

$$\frac{\Gamma \vdash t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \vdash c_i : M_i^{k_i} \quad \vdash e : \Gamma}{\vdash (t, c_1 \cdots c_n, e) : \star} \; \text{T-st}$$

By definition, $|q| = |e| + |c_1 \cdots c_n| = |e| + \sum_{i=1}^{n} |c_i|$. By Lemma 5.2, $|e| = |\Gamma|$ and $|c_i| = k_i$ for $1 \le i \le n$. Then $|q| = |\Gamma| + \sum_{i=1}^{n} k_i$. □

5.2 Soundness

Soundness is the fact that on typed states the Space KAM terminates. Here it is refined with our space analysis, showing that the weight of the type judgment is exactly the maximum space used by the run of the Space KAM.

The proof technique is mostly standard. It is based on a subject reduction property plus a space analysis of final states. What is slightly unusual is that subject reduction is not stated as an independent property, it is instead plugged into the proof of soundness. This is needed to prove the space bound.

Space Analysis of Final States. To prove that the weight system correctly measures the size of final states, we need an auxiliary lemma ensuring that closures and environments typed with empty types and dry type contexts induce null weight on their judgment.

Lemma 5.4 (Empty types and dry type environments induce null weight).

(1) *If* $\pi \triangleright \overset{w}{\vdash} e : \Gamma$ *and* Γ *is dry, then* $w = 0$.

(2) *If* $\pi \triangleright \overset{w}{\vdash} c : [\,]^k$, *then* $w = 0$.

Proof. By mutual induction on e and c.

(1) π has the following form:

$$\frac{\overset{w_x}{\vdash} e(x) : \Gamma(x)}{\overset{\max\{w_x | x \in \mathsf{dom}(e)\}}{\vdash} e : \Gamma} \; \text{T-env}$$

with $w = \max\{w_x | x \in \mathsf{dom}(e)\}$. Two cases:

(a) e is empty, i.e. $e = \epsilon$: then $w = 0$ and Γ is the empty type environment, for which $|\Gamma| = 0$. Therefore, we have $w = 0 = |\Gamma|$, validating the statement.

(b) e is not empty, i.e. $e \neq \epsilon$: Since Γ is dry, then $\overset{w_x}{\vdash} e(x) : [\,]^{k_x}$ for some k_x. By *i.h.* (point 2), $w_x = 0$. Then $w = \max\{w_x | x \in \mathsf{dom}(e)\} = 0$.

(2) π has the following form:

$$\frac{\overset{v}{\vdash} e : \Gamma \qquad \Gamma \overset{u}{\vdash} t : [\,]^k}{\overset{\max\{v,u\}}{\vdash} (t, e) : [\,]^k} \; \text{T-cl}$$

with $c = (t, e)$ and $w = \max\{v, u\}$. Since the typing of t comes necessarily from a rule T-none, Γ is dry, $u = 0$. Then we can apply the *i.h.* (point 1) to e, obtaining $v = 0$. Thus we have $w = 0$.

□

Proposition 5.5 (The weight of final states is their space). *Let q be a final state and $\pi \triangleright \vdash^{w} q : \star$. Then $w = |q|$.*

Proof. Final states have the shape $(\lambda x.t, e, \epsilon)$. Then π has the following shape:

$$\dfrac{\dfrac{\mathsf{dom}(\Gamma) = \mathsf{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \overset{|\Gamma|}{\vdash} \lambda x.t : \star}\,\text{T-}\lambda_\star \qquad \overset{v}{\vdash} e : \Gamma}{\overset{\max\{v, |\Gamma|\}}{\vdash} (\lambda x.t, e, \epsilon) : \star}\,\text{T-st}$$

with $w = \max\{v, |\Gamma|\}$. By Lemma 5.4.1, $v = 0$. Therefore, $w = |\Gamma|$ and so $w = |e| = |q|$. □

Soundness. Soundness shall be proved by induction on the size of type derivation, which is defined ignoring some typing rules, as follows.

Definition 5.6 (Type derivations size). The size $|\pi|$ of a type derivation π is its number of rules without counting rules T-many, T-none, T-cl, and T-env.

The subject reduction argument shall need the following auxiliary lemma.

Lemma 5.7 (Multi-set splitting).

(1) Terms*: let $\pi \triangleright \Gamma \overset{w}{\vdash} t : M^k \uplus N^k$ with $k = 1 + |\Gamma|$. Then there exists two type derivations $\pi_M \triangleright \Gamma_M \overset{w_M}{\vdash} t : M^k$ and $\pi_N \triangleright \Gamma_N \overset{w_N}{\vdash} t : N^k$ such that $\Gamma_M \# \Gamma_N$, $\Gamma = \Gamma_M \uplus \Gamma_N$, $|\pi| = |\pi_M| + |\pi_N|$ and $w = \max\{w_M, w_N\}$.*
(2) Closures*: let $\pi \triangleright \overset{w}{\vdash} c : M^k \uplus N^k$. Then there exists two type derivations $\pi_M \triangleright \overset{w_M}{\vdash} c : M^k$ and $\pi_N \triangleright \overset{w_N}{\vdash} c : N^k$ such that $|\pi| = |\pi_M| + |\pi_N|$ and $w = \max\{w_M, w_N\}$.*
(3) Environments*: let Γ and Δ summable and $\pi \triangleright \overset{w}{\vdash} e : \Gamma \uplus \Delta$. Then there exists two type derivations $\pi_\Gamma \triangleright \overset{w_\Gamma}{\vdash} e|_{\mathsf{dom}(\Gamma)} : \Gamma$ and $\pi_\Delta \triangleright \overset{w_\Delta}{\vdash} e|_{\mathsf{dom}(\Delta)} : \Delta$ such that $|\pi| = |\pi_\Gamma| + |\pi_\Delta|$ and $w = \max\{w_\Gamma, w_\Delta\}$.*

Proof.

(1) Suppose that $M^k = [\,]^k$. Then $M^k \uplus N^k = N^k$. Now, let Γ_M be defined as the unique dry type context such that $\mathsf{dom}(\Gamma_M) = \mathsf{dom}(\Gamma)$ and it is summable with Γ—note that necessarily $|\Gamma| = |\Gamma_M|$. By Lemma 5.1, $\mathsf{dom}(\Gamma) = \mathsf{fv}(t)$, and so we obtain the following derivation:

$$\dfrac{}{\Gamma_M \overset{0}{\vdash} t : [\,]^{1+|\Gamma_M|}}\,\text{T-none}$$

which is the π_M of the statement. We also take $\pi_N := \pi$, and the statement holds.
If $N^k = [\,]^k$ the proof is as in the previous case.
Assume now that both M^k and N^k are non-empty, say $M^k = [A_1, \ldots, A_m]^k$ and $N^k = [A_{m+1}, \ldots, A_n]^k$. Then π has the following shape:

$$\dfrac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \leq i \leq n \quad \#_i \Gamma_i}{\uplus_{i=1}^{n} \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_1, .., A_n]^{1+|\uplus_{i=1}^{n}\Gamma_i|}}\,\text{T-many}$$

and the two derivations π_M and π_N are obtained as follows

$$\pi_M := \frac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \le i \le m \quad \#_i\Gamma_i}{\uplus_{i=1}^m \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_1, .., A_m]^{1+|\uplus_{i=1}^m \Gamma_i|}}\ \text{T-MANY}$$

and

$$\pi_N := \frac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad m+1 \le i \le n \quad \#_i\Gamma_i}{\uplus_{i=m+1}^n \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_{m+1}, .., A_n]^{1+|\uplus_{i=m+1}^n \Gamma_i|}}\ \text{T-MANY}$$

which clearly satisfy the statement.

(2) If $c = (t, e)$ then π has the following form

$$\frac{\overset{w}{\vdash} e : \Gamma \qquad \Gamma \overset{v}{\vdash} t : M^k}{\overset{\max\{w,v\}}{\vdash} (t, e) : M^k}\ \text{T-CL}$$

Then the *i.h.* (point 1) applied to the right premise gives two derivations for t with respect to M^k and N^k. In particular it gives two summable type contexts Γ_M and Γ_N with which one applies the *i.h.* (point 3) to the left premise, obtaining two derivations for e with respect to Γ_M and Γ_N. Pairing the respective derivations for t and e one obtains the statement.

(3) If e is empty then both Γ and Δ are empty and the statement trivially holds. Otherwise, it follows from applying the *i.h.* (point 2) for each variable in $\text{dom}(\Gamma) \cap \text{dom}(\Delta)$.

□

Theorem 5.8 (Soundness). *Let q be a Space KAM reachable state such that there is a derivation $\pi \triangleright \vdash^w q : \star$. Then*

(1) Termination*: there is a run $\rho : q \rightarrow_{\text{SpKAM}}{}^* q_f$ to a final state. Moreover,*
(2) Space bound*: $w = |\rho|_{\text{sp}}$.*

Proof. The proof is by induction on $|\pi|$ and case analysis on whether q is final. If q is final then the run ρ in the statement is given by the empty run. Therefore, we have $|\rho|_{\text{sp}} = |q|$ and the required space bound becomes $w = |q|$, which is given by Prop. 5.5. If q is not final then $q \rightarrow_{\text{SpKAM}} q'$. By examining all the transition rules. We set $A := M_1^{k_1} \rightarrow \cdots \rightarrow M_n^{k_n} \rightarrow \star$ and $\pi := \pi_1 \cdots \pi_n$. We report just one case that illustrates the methodology. The other ones can be found in the associated technical report [Accattoli et al. 2022b].

Case $\rightarrow_{\text{sea}_v}$, i.e. $q = (tx, e, \pi)$ and $q' = (t, e|_t, e(x) \cdot \pi)$. The type derivation π typing q has the following shape:

$$\frac{\dfrac{\Gamma \overset{w'}{\vdash} t : M \rightarrow A}{\Gamma \uplus x : M \overset{w'}{\vdash} tx : A}\ \text{T-@}_2 \qquad \pi_e \triangleright \overset{u}{\vdash} e : \Gamma \uplus x : M \qquad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w', u, v_i\}}{\vdash} (tx, e, \pi) : \star}\ \text{T-ST}$$

with $w = \max_i\{w', v_i, u\}$. By Lemma 5.7, there are two derivations $\pi_\Gamma \triangleright \overset{u_\Gamma}{\vdash} e|_{\text{dom}(\Gamma)} : \Gamma$ and $\pi_{x:M} \triangleright \overset{u_{x:M}}{\vdash} e(x) : x : M$ such that $|\pi_e| = |\pi_\Gamma| + |\pi_{x:M}|$ and $u = \max\{u_\Gamma, u_{x:M}\}$. By the environment domain invariant (Lemma 3.1), $\text{dom}(\Gamma) = \text{fv}(t)$. Moreover, $\pi_{x:M}$ is necessarily the conclusion of a unary T-ENV rule of premise $\pi_M \triangleright \overset{u_M}{\vdash} e(x) : M$ for which $|\pi_M| = |\pi_{x:M}|$ and $u_M = u_{x:M}$. Then, q'

can be typed by the following derivation π':

$$\pi' := \cfrac{\Gamma' \overset{w'}{\vdash} t : M \to A \quad \pi_\Gamma \triangleright \overset{u_\Gamma}{\vdash} e|_t : \Gamma \quad \pi_M \triangleright \overset{u_M}{\vdash} e(x) : M \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w', u_\Gamma, u_M, v_i\}}{\vdash} (t, e|_t, e(x) \cdot \pi) : \star} \text{ T-ST}$$

Since $|\pi_e| = |\pi_\Gamma| + |\pi_{x:M}| = |\pi_\Gamma| + |\pi_M|$, we have $|\pi| > |\pi'|$ (beccause the T-@$_2$ rule is removed), and so we can apply the *i.h.*, obtaining a run $\sigma : q' \to_{\mathrm{SpKAM}}{}^* q_f$ to a final state such that $\max_i\{w', u_\Gamma, u_M, v_i\} = |\sigma|_{\mathrm{sp}}$. Then there is a run $\rho : q \to_{\mathrm{SpKAM}}{}^* q_f$, proving the first part of the statement (termination).

For the space bound, note that, since $u = \max\{u_\Gamma, u_{x:M}\} = \max\{u_\Gamma, u_M\}$, we have $w = \max_i\{w', u_\Gamma, u_M, v_i\}$. Additionally, $|q| \leq |q'|$, because by the environment domain invariant $e|_t$ removes at most $e(x)$ from e, which is however added to the stack. Then $|\rho|_{\mathrm{sp}} = |\sigma|_{\mathrm{sp}} = \max_i\{w', u_\Gamma, u_M, v_i\} = w$, proving the second part of the statement. □

As a corollary, we can transfer back the soundness result from typed *states* to typed *terms*.

Corollary 5.9 (Soundness, on Terms). *Let t be a closed λ-term. If there exists $\pi \triangleright \vdash^w t : \star$, then there exists a complete Space KAM run ρ from t such that $|\rho|_{\mathrm{sp}} = w$.*

5.3 Completeness

Completeness is the fact that all states on which the Space KAM terminates are typable. Here the proof technique is standard: we show that final states are typable, that a subject expansion property holds, and then we infer completeness. We do not perform any space analysis, since it already follows from the soundness part, once we know that a state is typable.

Final States are Typable. To prove that every final state is typable, we need an auxiliary lemma ensuring that closures and environments can always be typed with empty types and dry type contexts.

Lemma 5.10 (Closures and environments are typable).

(1) *There exists a dry type context Γ and a derivation $\pi \triangleright \vdash e : \Gamma$ for every environment e.*
(2) *There exist k and a derivation $\pi \triangleright \vdash c : [\,]^k$ for every closure c.*

Proof. By mutual induction on e and c.

(1) If e is empty then Γ is the empty type context. Otherwise, by *i.h.* (point 2), for every closure $e(x)$ with $x \in \mathrm{dom}(e)$ there exists k_x and a derivation $\pi \triangleright \vdash c : [\,]^k$. Then π is defined as follows:

$$\cfrac{\vdash e(x) : \Gamma(x)}{\vdash e : \Gamma} \text{ T-ENV}$$

(2) Let $c = (t, e)$. By the environments domain invariant (Lemma 3.1), $\mathrm{dom}(e) = \mathrm{fv}(t)$. By *i.h.* (point 1), there exists a dry type context Γ and a derivation $\pi \triangleright \vdash e : \Gamma$. Since $\mathrm{dom}(e) = \mathrm{fv}(t)$, we can apply rule T-none deriving a judgment $\Gamma \vdash t : [\,]^{1+|\Gamma|}$. Then π is defined as follows:

$$\cfrac{\vdash e : \Gamma \qquad \Gamma \vdash t : [\,]^{1+|\Gamma|}}{\vdash (t, e) : [\,]^{1+|\Gamma|}} \text{ T-CL}$$

□

Proposition 5.11 (Final states are typable). *Let q be a final state. Then there exists a type derivation $\pi \triangleright \vdash q : \star$.*

Proof. Final states have the shape $(\lambda x.t, e, \epsilon)$. By Lemma 5.10.1, there is a type derivation $\vdash e : \Gamma$ with Γ dry. By the environments domain invariant (Lemma 3.1), $\mathrm{dom}(e) = \mathrm{fv}(\lambda x.t)$. Then π is defined as follows:

$$\dfrac{\dfrac{\mathrm{dom}(\Gamma) = \mathrm{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \vdash \lambda x.t : \star}\,\text{T-}\lambda_\star \qquad \vdash e : \Gamma}{\vdash (\lambda x.t, e, \epsilon) : \star}\,\text{T-ST}$$

□

Subject Expansion. The proof of subject expansion is standard. There is only one delicate point, in expanding the application transitions $\to_{\mathrm{sea}_{\neg v}}$ and $\to_{\mathrm{sea}_v}$, where one needs to ensure that the two type contexts for the premises of the application are summable.

Proposition 5.12 (Subject expansion). *If $q \to_{\mathrm{SpKAM}} q'$ and there exists $\pi' \triangleright \vdash q' : \star$, then there exists $\pi \triangleright \vdash q : \star$.*

Proof. There are three cases that require something more than simply reading backwards the subject reduction argument in the proof of the correctness theorem. One is $\to_{\beta_w}$, where in addition one needs to type the garbage collected closure, but this is ensured by Lemma 5.10.1. The other two are the application transitions $\to_{\mathrm{sea}_{\neg v}}$ and $\to_{\mathrm{sea}_v}$. We treat $\to_{\mathrm{sea}_{\neg v}}$, the case of $\to_{\mathrm{sea}_v}$ is analogous. We have $q = (tu, e, c_1 \cdots c_n) \to_{\mathrm{sea}_{\neg v}} (t, e|_t, (u, e|_u) \cdot c_1 \cdots c_n) = q'$. The type derivation π' typing q' has the following shape:

$$\pi' = \dfrac{\Gamma \vdash t : M^k \to A \quad \vdash e|_t : \Gamma \quad \dfrac{\Delta \vdash u : M^k \qquad \vdash e|_u : \Delta}{\vdash (u, e|_u) : M^k}\,\text{T-CL} \quad \vdash c_i : M_i^{k_i}}{\vdash (t, e|_t, (u, e|_u) \cdot c_1 \cdots c_n) : \star}\,\text{T-ST}$$

Assuming that $\Gamma \# \Delta$, that we shall prove below, the type derivation π for q is given by:

$$\pi := \dfrac{\dfrac{\Gamma \vdash t : M^k \to A \quad \Delta \vdash u : M^k \quad \Gamma \# \Delta}{\Gamma \uplus \Delta \vdash tu : A}\,\text{T-@}_1 \quad \vdash e : \Gamma \uplus \Delta \quad \vdash c_i : M_i^{k_i}}{\vdash (tu, e, c_1 \cdots c_n) : \star}\,\text{T-ST}$$

where the derivation for $\vdash e : \Gamma \uplus \Delta$ is obtained by an omitted and straightforward *multi-sets merging* lemma dual to the multi-sets splitting lemma (Lemma 5.7) used for correctness.

We now prove $\Gamma \# \Delta$, which is the additional bit not present in the proof of subject reduction. Let $x \in \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta)$. Then $e_t(x) = e_u(x) = e(x) = c$ for some closure c. By Lemma 5.2.2, we have both $\Gamma(x) = N_1^{|c|}$ and $\Delta(x) = N_2^{|c|}$ for some multi types N_1 and N_2, that is, $\Gamma \# \Delta$.

□

Theorem 5.13 (Completeness). *Let q a reachable state and $\rho : q \to_{\mathrm{SpKAM}}^* q_f$ be a KAM run to a final state q_f. Then there exist $\pi \triangleright \vdash q : \star$.*

Proof. By induction on the length $|\rho|$ of ρ. If $|\rho| = 0$ then $q = q_f$, and the existence of a typing derivation for q is given by Prop. 5.11. If $|\rho| > 0$ then ρ is given by a transition $q \to_{\mathrm{SpKAM}} q'$ followed by an execution $\sigma : q' \to_{\mathrm{SpKAM}}^* q_f$. By *i.h.*, we obtain $\pi \triangleright \vdash q' : \star$, and by subject expansion (Prop. 5.12) we obtain $\pi \triangleright \vdash q : \star$. □

Again, as a corollary, we can transfer the completeness property from *states* to *terms*.

Corollary 5.14 (Completeness, on Terms). *If t is a closed λ-term such that there exists a complete Space KAM run from t, then there exists $\pi \triangleright \vdash t : \star$.*

Putting together quantitative soundness and completeness gives the quantitative correctness of the closure type system.

Theorem 5.15 (Correctness). *Let t be a closed λ-term. Then there exists a complete Space KAM run ρ from t such that $|\rho|_{\mathsf{sp}} = w$ if and only if there exists $\pi \triangleright \vdash^{w} t : \star$.*

6 CAPTURING THE SPACE KAM LOW-LEVEL TIME

Low-Level Time. Accattoli et al. [2022a] show that the Space KAM run on a closed term t cannot be implemented on Turing machines with an overhead in time which is polynomial in the number n of weak head reduction steps from t (which is the natural cost model for time in Closed CbN), nor in the number of Space KAM steps (which is linear in n). This is because of the lack of sharing of the Space KAM data structures. Indeed, in transitions $\rightarrow_{\mathsf{sea}_{\neg\mathsf{v}}}$ and $\rightarrow_{\mathsf{sea}_{\neg\mathsf{v}}}$ the environment, of a priori unbounded size, can be duplicated. Of course, this copy operation cannot take a constant amount of time, and—as Accattoli et al. [2022a] shows—it can actually lead to an exponential overhead.

It is however possible to adopt another more low-level time measure, the *real* implementation time, that is, the time actually taken by the Space KAM to run. One transition does not count anymore for one unit of time, but we have to take into account the amount of time needed to manage the involved data structures. Since we do not rest on any sharing mechanism, roughly the time consumed by any transition is proportional to the sizes of the involved states[6]. Thus, in analogy to what we have done for space we define the low-level time consumption of the Space KAM in the following way.

Definition 6.1 (Low-Level Run Time). Let $\rho : \mathsf{init}(t_0) \rightarrow_{\mathsf{SpKAM}}{}^{*} q$ be a Space KAM run. Then the low-level time consumption of the run ρ is defined as follows:

$$|\rho|_{\mathsf{ti}} := \sum_{q' \in \rho} |q'|$$

Although this low-level time measure can be exponential in the number of β-steps a term needs to normalize, we have the following reasonability result[7].

Theorem 6.2 ([Accattoli et al. 2022a], The Space KAM Is Reasonable for Low-Level Time). *Closed CbN evaluation $\rightarrow_{wh}$ and the low-level time of the Space KAM $|\cdot|_{\mathsf{ti}}$ provide a reasonable time cost model for the λ-calculus.*

The New Weighting System. We now turn to the definition of a weighted type system that is able to capture the Space KAM low-level time. Actually, the type system is the same of section 4, what changes is just the weight assignment, that can be found in Fig. 7. Essentially, all the max have been turned into sums. More precisely, one can observe that in every rule (except for T-none and T-many), the weight w of the conclusion with judgment $\Gamma \vdash^{w} t : A$ is exactly the sum of all the weights of the premises *plus* $|\Gamma|$ and $|A|$, i.e. the size of the Space KAM state corresponding to the judgment in the conclusion.

Soundness. The proof of soundness proceeds exactly as the one in Section 5. This is why we report only the main ingredients, leaving the technical details to the associated technical report [Accattoli et al. 2022b]. Also in this case we have to extend the type system to the machine components (in Fig. 7), so that subject reduction/soundness is proved on states, rather than terms.

[6]Since we are interested in measure time up to a *polynomial* overhead, we are allowed to make some simplifications. For example, in this case, we do not discuss about the size of the input pointers, because it can be absorbed in the polynomial overhead.

[7]Intuitively, reasonability holds because in the simulation of Turing machines (the reasonable reference model), i.e. in the execution of the encoding of Turing machines into the λ-calculus, the exponential gap does not show up.

$$\frac{}{x : [A]^k \overset{k+|A|}{\vdash} x : A}\ \text{T-Var} \qquad \frac{\mathsf{dom}(\Gamma) = \mathsf{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \overset{|\Gamma|}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma, x : M^k \overset{w}{\vdash} t : A}{\Gamma \overset{w+|\Gamma|+|A|+k}{\vdash} \lambda x.t : M^k \to A}\ \text{T-}\lambda_1 \qquad \frac{\Gamma \overset{w}{\vdash} t : A \quad x \notin \mathsf{dom}(\Gamma)}{\Gamma \overset{w+|\Gamma|+|A|+k}{\vdash} \lambda x.t : [\,]^k \to A}\ \text{T-}\lambda_2$$

$$\frac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \le i \le n \quad \#_i \Gamma_i}{\uplus_{i=1}^n \Gamma_i \overset{\sum_i \{v_i\}}{\vdash} t : [A_1..A_n]^{1+|\uplus_{i=1}^n \Gamma_i|}}\ \text{T-many} \qquad \frac{\mathsf{dom}(\Gamma) = \mathsf{fv}(t) \quad \Gamma \text{ is dry}}{\Gamma \overset{0}{\vdash} t : [\,]^{1+|\Gamma|}}\ \text{T-none}$$

$$\frac{\Gamma \overset{w}{\vdash} t : M^k \to A \quad \Delta \overset{v}{\vdash} u : M^k \quad \Gamma \# \Delta}{\Gamma \uplus \Delta \overset{w+v+|\Gamma\uplus\Delta|+|A|}{\vdash} tu : A}\ \text{T-@}_1 \qquad \frac{\Gamma \overset{w}{\vdash} t : M^k \to A \quad \Gamma \# x : M^k}{\Gamma \uplus x : M^k \overset{w+|\Gamma|+k+|A|}{\vdash} tx : A}\ \text{T-@}_2$$

$$\frac{\overset{w_x}{\vdash} e(x) : \Gamma(x) \quad \forall x \in \mathsf{dom}(e)}{\overset{\sum\{w_x | x \in \mathsf{dom}(e)\}}{\vdash} e : \Gamma}\ \text{T-env} \qquad \frac{\overset{w}{\vdash} e : \Gamma \quad \Gamma \overset{v}{\vdash} t : M^k}{\overset{w+v}{\vdash} (t, e) : M^k}\ \text{T-cl}$$

$$\frac{\Gamma \overset{w}{\vdash} t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \overset{u}{\vdash} e : \Gamma \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{w+u+\sum_i v_i}{\vdash} (t, e, \pi) : \star}\ \text{T-st}$$

Fig. 7. Typing rules for terms (above) and machine components (below).

Time Analysis of Final States. We need an auxiliary lemma that allows us to prove that the weight system correctly measures the size of final states.

Lemma 6.3 (The weight of final states is their low-level time). *Let q be a final state and $\pi \triangleright \vdash^w q : \star$. Then $w = |q|$.*

Then we able to state the soundness theorem, this time for the low-level time weighting system.

Theorem 6.4 (Low-Level Time Soundness). *Let q be a Space KAM reachable state such that there is a derivation $\pi \triangleright \vdash^w q : \star$. Then the run $\rho : q {\to_{\mathrm{SpKAM}}}^* q_f$ to the final state is such that $w = |\rho|_{\mathrm{ti}}$*[8].

Proof. The proof is by induction on $|\pi|$ and case analysis on whether q is final. If q is final then the run ρ in the statement is given by the empty run, containing only q. Therefore, we have $|\rho|_{\mathrm{ti}} = |q|$ and the required time bound becomes $w = |q|$, which is given by Lemma 6.3. If q is not final then $q \to_{\mathrm{SpKAM}} q'$. By examining all the transition rules. We set $A := M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star$ and $\pi := \pi_1 \cdots \pi_n$.

We report just one case to illustrate the methodology, the ones can be found in the associated technical report [Accattoli et al. 2022b].

[8]We have already proved the soundness of the type system w.r.t. termination, which does not depend on the weighting system. This is why termination is not part of the statement.

Case $\to_{\text{sub}}$, i.e. $q = (x, [x{\leftarrow}(u, e)], \pi)$ and $q' = (u, e, \pi)$. The type derivation π typing q has the following shape:

$$\dfrac{\dfrac{}{x : [A]^k \overset{k+|A|}{\vdash} x : A}\;\textsc{T-Var} \qquad \dfrac{\dfrac{\Gamma \overset{w'}{\vdash} u : [A]^k \quad \overset{u}{\vdash} e : \Gamma}{\overset{u+w'}{\vdash} (u, e) : [A]^k}\;\textsc{T-cl}}{\overset{u+w'}{\vdash} [x{\leftarrow}(u, e)] : x : [A]^k}\;\textsc{T-env} \qquad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{k+|A|+u+w'+\sum_i v_i}{\vdash} (x, [x{\leftarrow}(u, e)], \pi) : \star}\;\textsc{T-st}$$

with $w = k + |A| + u + w' + \sum_i v_i$. The target state q' can be typed by the following derivation π':

$$\pi' := \dfrac{\Gamma \overset{w'}{\vdash} u : A \quad \overset{u}{\vdash} e : \Gamma \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{w'+u+\sum_i v_i}{\vdash} (u, e, \pi) : \star}\;\textsc{T-st}$$

Since the T-Var rule is removed, we have $|\pi| > |\pi'|$, and so we can apply the *i.h.*, obtaining a run $\sigma : q' \to_{\text{SpKAM}}{}^{*} q_f$ to a final state such that $w' + u + \sum_i v_i = |\sigma|_{\text{ti}}$.

For the time bound, note that $|\rho|_{\text{ti}} = |q| + |\sigma|_{\text{ti}} = w$. □

As a corollary, we can transfer back the soundness result from typed *states* to typed *terms*.

Corollary 6.5 (Low-Level Time Soundness, on Terms). *Let t be a closed λ-term. If there exists $\pi \triangleright \vdash^{w} t : \star$, then there exists a complete Space KAM run ρ from t such that $|\rho|_{\text{ti}} = w$.*

Putting together low-level time soundness and the completeness result of the previous section (which does not rests on the weighting system) one has the low-level time correctness of the time variant of the closure type system.

Theorem 6.6 (Low-Level Time Correctness). *Let t be a closed λ-term. Then there exists a complete Space KAM run ρ from t such that $|\rho|_{\text{ti}} = w$ if and only if there exists $\pi \triangleright \vdash^{w} t : \star$.*

Example 6.7. For the sake of completeness, we provide the closure type derivation of our example term $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$ decorated with our new weighting system, characterizing low-level time.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\star]^1 \vdash^{1} x : \star}\;\textsc{T-Var}}{x : [\star]^1 \vdash^{5} \lambda z.x : [\,]^3 \to \star}\;\textsc{T-}\lambda_2 \qquad \dfrac{}{x : [\,]^1, y : [\,]^1 \vdash^{0} xy : [\,]^3}\;\textsc{T-none}}{x : [\star]^1, y : [\,]^1 \vdash^{7} (\lambda z.x)(xy) : \star}\;\textsc{T-@}_1}{x : [\star]^1 \vdash^{9} \lambda y.(\lambda z.x)(xy) : [\,]^1 \to \star}\;\textsc{T-}\lambda_1}{x : [\star]^1 \vdash^{10} (\lambda y.(\lambda z.x)(xy))x : \star}\;\textsc{T-@}_2}{\vdash^{11} \lambda x.(\lambda y.(\lambda z.x)(xy))x : [\star]^1 \to \star}\;\textsc{T-}\lambda_1 \qquad \dfrac{\dfrac{}{\vdash^{0} \mathsf{I} : \star}\;\textsc{T-}\lambda_\star}{\vdash^{0} \mathsf{I} : [\star]^1}\;\textsc{T-many}}{\vdash^{11} (\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I} : \star}\;\textsc{T-@}_1$$

The final weight, i.e. the low-level time of the evaluation on the Space KAM is 11. One could easily check that the same result holds if one looks directly at the Space KAM execution.

7 CONCLUSIONS

This paper is set in the recent trend aiming at analyzing complexity properties of λ-terms through the use of multi types (a.k.a. non-idempotent intersection types). Taking advantage of a recent result about a reasonable space measure for the call-by-name λ-calculus, we craft a multi type system able to reflect such a measure on typed λ-terms.

There are at least two interesting directions for future work. Firstly, we would like to adapt the type system to cover call-by-value evaluation, where a notion of reasonable space measure is known [Accattoli et al. 2022a], too. Secondly, it would be interesting to understand what happens in the case of infinite computations. In these cases, the space consumption can be either finite, if the machine cycles, or infinite. In the former case, a newly devised (coinductive) type system could give the space bound. In the latter case, one could be interested in asymptotic properties, e.g. "during the (infinite) execution, the space consumption is logarithmic in time". This kind of properties could be useful in applications that have to do with infinite data, such as stream processing, and in general with lazy infinite (coinductive) data structures. For designing the type system, one might possibly build on Vial's study of infinitary evaluation based on a variant of multi types [Vial 2017].

REFERENCES

Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. 2000. Full Abstraction for PCF. *Inf. Comput.* 163, 2 (2000), 409–470. https://doi.org/10.1006/inco.2000.2930

Beniamino Accattoli. 2017. (In)Efficiency and Reasonable Cost Models. In *12th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2017 (ENTCS, Vol. 338)*. Elsevier, 23–43. https://doi.org/10.1016/j.entcs.2018.10.003

Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. 2014. Distilling abstract machines. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional progranitarmming, Gothenburg, Sweden, September 1-3, 2014*, Johan Jeuring and Manuel M. T. Chakravarty (Eds.). ACM, 363–376. https://doi.org/10.1145/2628136.2628154

Beniamino Accattoli and Claudio Sacerdoti Coen. 2017. On the value of variables. *Inf. Comput.* 255 (2017), 224–242.

Beniamino Accattoli and Ugo Dal Lago. 2016. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science* 12, 1 (2016). https://doi.org/10.2168/LMCS-12(1:4)2016

Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2020a. The Machinery of Interaction. In *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming* (Bologna, Italy) *(PPDP '20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 15 pages. https://doi.org/10.1145/3414080.3414108

Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2021a. The (In)Efficiency of Interaction. *Proc. ACM Program. Lang.* 5, POPL, Article 51 (Jan. 2021), 33 pages. https://doi.org/10.1145/3434332

Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2021b. The Space of Interaction. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–13. https://doi.org/10.1109/LICS52264.2021.9470726

Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2022a. Reasonable Space for the λ-calculus, Logarithmically. In *2022 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. https://doi.org/10.1145/3531130.3533362

Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2018. Tight Typings and Split Bounds. *Proc. ACM Program. Lang.* 2, ICFP (2018), 94:1–94:30.

Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2020b. Tight typings and split bounds, fully developed. *J. Funct. Program.* 30 (2020), e14. https://doi.org/10.1017/S095679682000012X

Beniamino Accattoli and Giulio Guerrieri. 2018. Types of Fireballs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11275)*, Sukyoung Ryu (Ed.). Springer, 45–66. https://doi.org/10.1007/978-3-030-02768-1_3

Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. 2019. Types by Need. In *28th European Symposium on Programming, ESOP 2019,Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11423)*. Springer, 410–439. https://doi.org/10.1007/978-3-030-17184-1_15

Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2022b. Multi Types and Reasonable Space (Long Version). *CoRR* (2022). https://doi.org/10.48550/arXiv.2207.08795

Sandra Alves, Delia Kesner, and Daniel Ventura. 2019. A Quantitative Understanding of Pattern Matching. In *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway.* 3:1–3:36. https://doi.org/10.4230/LIPIcs.TYPES.2019.3

Alexis Bernadet and Stéphane Graham-Lengrand. 2013. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science* 9, 4 (2013). https://doi.org/10.2168/LMCS-9(4:3)2013

Guy E. Blelloch and John Greiner. 1995. Parallelism in Sequential Functional Languages. In *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995.* ACM, 226–237. https://doi.org/10.1145/224164.224210

Viviana Bono and Mariangiola Dezani-Ciancaglini. 2020. A tale of intersection types. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 7–20. https://doi.org/10.1145/3373718.3394733

Antonio Bucciarelli and Thomas Ehrhard. 2001. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Log.* 109, 3 (2001), 205–241. https://doi.org/10.1016/S0168-0072(00)00056-7

Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. 2020. The Bang Calculus Revisited. In *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12073)*, Keisuke Nakano and Konstantinos Sagonas (Eds.). Springer, 13–32. https://doi.org/10.1007/978-3-030-59025-3_2

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. https://doi.org/10.1093/jigpal/jzx018

Mario Coppo and Mariangiola Dezani-Ciancaglini. 1978. A new type assignment for λ-terms. *Arch. Math. Log.* 19, 1 (1978), 139–156. https://doi.org/10.1007/BF02011875

Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. 2021. Intersection types and (positive) almost-sure termination. *Proc. ACM Program. Lang.* 5, POPL, Article 32 (2021), 32 pages. https://doi.org/10.1145/3434313

Vincent Danos, Hugo Herbelin, and Laurent Regnier. 1996. Game Semantics & Abstract Machines. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996.* IEEE Computer Society, 394–405. https://doi.org/10.1109/LICS.1996.561456

Vincent Danos and Laurent Regnier. 1999. Reversible, irreversible and optimal lambda-machines. *Theoretical Computer Science* 227, 1 (1999), 79–97. https://doi.org/10.1016/S0304-3975(99)00049-3

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

Daniel de Carvalho. 2018. Execution time of λ-terms via denotational semantics and intersection types. *Math. Str. in Comput. Sci.* 28, 7 (2018), 1169–1203. https://doi.org/10.1017/S0960129516000396

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. https://doi.org/10.1016/j.tcs.2010.12.017

Mariangiola Dezani-Ciancaglini, Paola Giannini, and Betti Venneri. 2018. Intersection Types in Java: Back to the Future. In *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday (Lecture Notes in Computer Science, Vol. 11200)*, Tiziana Margaria, Susanne Graf, and Kim G. Larsen (Eds.). Springer, 68–86. https://doi.org/10.1007/978-3-030-22348-9_6

Yannick Forster, Fabian Kunze, and Marc Roth. 2020. The weak call-by-value λ-calculus is reasonable for both time and space. *Proc. ACM Program. Lang.* 4, POPL (2020), 27:1–27:23. https://doi.org/10.1145/3371095

Timothy S. Freeman and Frank Pfenning. 1991. Refinement Types for ML. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991*, David S. Wise (Ed.). ACM, 268–277. https://doi.org/10.1145/113445.113468

Daniel P. Friedman, Abdulaziz Ghuloum, Jeremy G. Siek, and Onnie Lynn Winebarger. 2007. Improving the lazy Krivine machine. *High. Order Symb. Comput.* 20, 3 (2007), 271–293.

Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. 2008. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *J. ACM* 55, 4 (2008), 19:1–19:64. https://doi.org/10.1145/1391289.1391293

Jean-Yves Girard. 1989. Geometry of Interaction 1: Interpretation of System F. In *Logic Colloquium '88*, R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 127. Elsevier, 221 – 260. https://doi.org/10.1016/S0049-237X(08)70271-4

Delia Kesner, Loïc Peyrot, and Daniel Ventura. 2021. The Spirit of Node Replication. In *FoSSaCS (Lecture Notes in Computer Science, Vol. 12650)*. Springer, 344–364.

Delia Kesner and Pierre Vial. 2020. Consuming and Persistent Types for Classical Logic. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 619–632. https://doi.org/10.1145/3373718.3394774

Delia Kesner and Andrés Viso. 2022. Encoding Tight Typing in a Unified Framework. In *CSL (LIPIcs, Vol. 216)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 27:1–27:20.

Jean-Louis Krivine. 2007. A Call-by-name Lambda-calculus Machine. *Higher Order Symbol. Comput.* 20, 3 (2007), 199–207. https://doi.org/10.1007/s10990-007-9018-9

Stéphane Lengrand, Pierre Lescanne, Daniel J. Dougherty, Mariangiola Dezani-Ciancaglini, and Steffen van Bakel. 2004. Intersection types for explicit substitutions. *Inf. Comput.* 189, 1 (2004), 17–42. https://doi.org/10.1016/j.ic.2003.09.004

Ian Mackie. 1995. The Geometry of Interaction Machine. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, Ron K. Cytron and Peter Lee (Eds.). ACM Press, 198–208. https://doi.org/10.1145/199448.199483

Peter Møller Neergaard and Harry G. Mairson. 2004. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, Chris Okasaki and Kathleen Fisher (Eds.). ACM, 138–149. https://doi.org/10.1145/1016850.1016871

Benjamin C. Pierce. 1997. Intersection Types and Bounded Polymorphism. *Math. Struct. Comput. Sci.* 7, 2 (1997), 129–193. https://doi.org/10.1017/S096012959600223X

David Sands, Jörgen Gustavsson, and Andrew Moran. 2002. Lambda Calculi and Linear Speedups. In *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday] (Lecture Notes in Computer Science, Vol. 2566)*, Torben Æ. Mogensen, David A. Schmidt, and Ivan Hal Sudborough (Eds.). Springer, 60–84. https://doi.org/10.1007/3-540-36377-7_4

Peter Sestoft. 1997. Deriving a Lazy Abstract Machine. *J. Funct. Program.* 7, 3 (1997), 231–264.

Cees F. Slot and Peter van Emde Boas. 1988. The Problem of Space Invariance for Sequential Machines. *Inf. Comput.* 77, 2 (1988), 93–122. https://doi.org/10.1016/0890-5401(88)90052-1

Steffen van Bakel. 1992. Complete Restrictions of the Intersection Type Discipline. *Theor. Comput. Sci.* 102, 1 (1992), 135–163. https://doi.org/10.1016/0304-3975(92)90297-S

Peter van Emde Boas. 1990. Machine Models and Simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*. Elsevier and MIT Press, 1–66.

Pierre Vial. 2017. Infinitary intersection types as sequences: A new answer to Klop's problem. In *LICS*. IEEE Computer Society, 1–12.

Mitchell Wand. 2007. On the correctness of the Krivine machine. *High. Order Symb. Comput.* 20, 3 (2007), 231–235.