The Collapse of Sequential to Multiset Intersection Type Systems is Surjective

IRIF and Université Paris Diderot

pvial@irif.fr

Pierre Vial

We show that every (finite or not) typing derivation of system $\mathscr{R}$, using non-idempotent intersection, which is the infinitary version of de Carvalho's system $\mathscr{R}_0$, can be represented in a *rigid*, non-idempotent intersection type system S. Namely, whereas non-idempotent intersection is represented by multisets in system $\mathscr{R}$, system S resort to families of types indexed by integers, called *tracks*. The *rigidity* is here related to the fact that those indexes matter as well as the order in which the types are quoted in a family of types. Sequential constructions naturally collapse on multisets constructions and any S-derivation easily collapses on a $\mathscr{R}$-derivation.

System S has more fine-grained features that those of $\mathscr{R}$, that allow us to answer some theoretical questions in *infinitary* frameworks, but the use of sequential constructions make the rule typing the application very syntactical compared to that of System $\mathscr{R}$. It is not obvious that, conversely, the $\mathscr{R}$-derivation may be lifted into a S *i.e.* if we can rewrite any multiset-based derivation into a sequential derivations. If it were not, System S would be more fine-grained but having a narrower span than that of System $\mathscr{R}$.

We prove that indeed, every $\mathscr{R}$-derivation is the collapse of a S-derivations. For that, we consider an intermediate type system, featuring labelled trees as types as S does, but in which the application rule is relaxed by the use of isomorphisms of labelled trees encoding the rewriting of multiset types. We prove that every intermediate derivation using complex isomorphisms is itself isomorphic to a derivation of S, that uses only identity isomorphisms between trees. The main technical contribution of this paper is managing to work in a coinductive framework that in which no kind of productivity is ensured by some guard condition.

1 Introduction

1.1 Types Systems and Normalization

An important dynamical property of λ-terms is *normalization*: for instance, a term is **Head Normalizing (HN)** if it can be reduced to a Head Normal Form (*i.e.* a term of the form $\lambda x_1 \dots x_p . x t_1 \dots t_q$) and a term is **Weakly Normalizing (WN)** when it can be reduced to a **Normal Form** (*i.e.* a term without redex). A term t is **Strongly Normalizing** if there is not infinite reduction paths starting from t.

Simple type systems for λ-calculus were introduced by Church. Typing consists in assigning formulas called *types* to the variables of a term t and then computing the type of t while checking that the *typing rules* (of the system) are respected. These typing rules usually emulate Natural Deduction. For instance, if t and u respectively have types $A \to B$ and A, then $t\,u$ will have type B (*modus ponens*). Turing and Curry proved that, if a term is typable in the original system (that we call `Simple`), then it is normalizing.

However, it is not difficult to find normalizing terms – even normal forms – that are not typable. First, it is impossible to type $t\,u$ in `Simple` if t and u have the same type (the equality $A = A \to B$ is impossible).

To appear in EPTCS.

Thus, it is impossible to type the normal form xx or the term $\Delta = \lambda x.xx$.

Intersection Type Systems (ITS), introduced by Coppo and Dezani [5] are used to statically *characterize* classes of normalizing terms inside different calculi. In contrast with `Simple`, in an ITS, we will have: for all term t, t is normalizing iff it is typable. The main feature of an ITS is the following: each time we meet an occurrence of a variable x, we may assign to it a new type. For instance, xx becomes easily typable (if x receives the types A and $A \to B$, then we can type xx with B). Thus, intersection type allows a kind of *polymorphism*.

The types that are assigned to a variable are usually "collected" by an intersection operator $\wedge$. Here, this operator on the set of types will always be assumed to be associative and commutative and to have a neutral element $1_\wedge$. Let us discuss now the possible idempotency of this operator.

1.2 Idempotent and Non-Idempotent Intersections

The operator $\wedge$ is said to be **idempotent** if for all type A, $A \wedge A = A$.

The original type systems feature *idempotent* intersections. In that case, the types $A \wedge B \wedge A$ and $A \wedge B$ are equal.

Let us call **strict** (after van Bakel [9]) a type A that is not an intersection *i.e.* A type that is not equal $B \wedge C$ for some B, $C \neq 1_\wedge$ (thus, a strict type is a type that is atomic for the intersection operator). If the A_i and the B_j are strict, by associativity, commutativity, idempotency, we have $\wedge_{i\in I} A_i = \wedge_{j\in J} B_j$ iff $\{A_i\}_{i\in I} = \{B_j\}_{j\in J}$. Thus, the intersection $\wedge_{i\in I} A_i$ of strict types A_i may be written $\{A_i\}_{i\in I}$ and called a **set type**. Moreover, the intersection $\{A_i\}_{i\in I} \wedge \{B_j\}_{j\in J}$ of two set types is $(\wedge_{i\in I} A_i) \wedge (\wedge_{j\in J} B_j)$ *i.e.* it is $\{A_i\}_{i\in I} \cup \{B_j\}_{j\in J}$. Thus, in the idempotent case, the intersection operator corresponds to the set-theoretic union!

Gardner and de Carvalho [7, 4] provided a new characterization of the set of HN terms by means of a type system $\mathscr{R}_0$, which resorted to *non-idempotent* intersection types. This framework allowed to replace Tait's Realizability Argument – used to prove the implication "Typable $\Rightarrow$ Normalizable" – by a considerably simpler, arithmetical one.

Concretely, non-idempotency means here that types occur with a multiplicity in an intersection type. Namely, if the σ_i and the τ_j are strict, we have $\wedge_{i\in I} \sigma_i = \wedge_{j\in J} \tau_j$ iff $[\sigma_i]_{i\in I} = [\tau_j]_{j\in J}$. The intersection of strict types $\sigma_1, \ldots, \sigma_n$ may be represented by the *multiset* $[\sigma_1, \ldots, \sigma_n]$: the order of this enumeration does not matter, but the number of occurrences of a type does, *e.g.* we have $[\sigma, \tau, \sigma] = [\sigma, \sigma, \tau] \neq [\sigma, \tau]$. In system $\mathscr{R}_0$, the assignments $x : [\sigma, \tau, \sigma]$ and $x : [\sigma, \tau]$ cannot be interchanged, and the application rule *accumulates* the typing information in the environments: if x is assigned $[\sigma_1, \sigma_2, \sigma_3]$, $[\sigma_2, \sigma_4]$ and $[\sigma_4]$ in the premises of an application rule, then it will be assigned their **multiset sum** *i.e.* $[\sigma_1, \sigma_2, \sigma_2, \sigma_3, \sigma_4, \sigma_4]$ in its conclusion.

Thus, in system $\mathscr{R}_0$, a type can be regarded as a *resource*, which the quantitative argument proving that typability head-normalizability relies on: when a *typed* redex is reduced inside a derivation Π, it yields a new derivation Π' (typing the reduced term) while strictly decreasing a positive integer measure. It means that, at some point, there are no more typed redexes, so that the reduced is a (partial) NF.

1.3 Type Characterization in an Infinitary Framework

System $\mathscr{R}_0$ drew in the last year a great interest towards quantitative – *resource aware* – type systems and it paved the road for many works [1, 2, 3], either simplifying previous results or establishing new

ones.

In [12], we investigated the possibility of a type characterization of weak normalizability in Λ^{001}, an infinitary λ-calculus which was introduced in [8].

The finitary type system $\mathscr{R}_0$ can be given an infinitary variant $\mathscr{R}$ by taking its rules *coinductively* (instead of *inductively*) and allowing multisets to be infinite. We may notice $\mathscr{R}$ allows irrelevant infinite derivation *e.g.* some non-head normalizing terms are typable in $\mathscr{R}$. This observation suggested to use a validity criterion relying on the notion *approximability* to discard irrelevant proofs. However, we showed that this notion of approximability could not be formulated in $\mathscr{R}$, roughly because it is not possible to distinguish two occurrences of the same type in a multiset (see next section). This led us to resort to rigid constructions.

1.4 Rigid Non-Idempotent Intersections

In order to be able to characterize the of WN terms in an infinitary λ-calculus, we used an infinitary type assignment system S with non-idempotent intersections [12].

System S differs from the infinitary version $\mathscr{R}$ of the finitary type system $\mathscr{R}_0$ in that, the intersections are **rigid** in S: the *multisets* of types (called **multiset types**) used in $\mathscr{R}$ are (coinductively) replaced by *families* of types indexed by integers (those integers are called **tracks**). Such a family of types is called a **sequence type**. For instance, the sequence type $(T_k)_{k\in\{2,3,8\}}$, with $T_2 = S$, $T_3 = T$ and $T_8 = S$, is an intersection of two occurrences of type S (placed on tracks 2 and 8) and one occurrence of type T (placed on track 3). This sequence type is also denoted $(2{\cdot}S, 3{\cdot}T, 8{\cdot}S)$. Notice that here, we have the "pointers" 2 and 8 to distinguish the two occurrences of S. We have a "disjoint union" operator $\uplus$ for sequences *e.g.* $(2{\cdot}S, {\cdot}7) \uplus (8{\cdot}S) = (2{\cdot}S, 3{\cdot}T, 8{\cdot}S)$. In constrast, with multisets, we have $[\sigma,\tau]+[\sigma] = [\sigma,\sigma,\tau]$, but, in this equality, we have no way to relate one occurrence of σ in $[\sigma,\sigma,\tau]$ to $[\sigma,\tau]$ rather than $[\sigma]$ and *vice versa*.

To be equal, two S-types need to be *syntactically* equal – let us say informally that the equality is **tight** in S – *e.g.* $(T_k)_{k\in\{2,3,8\}} \neq (T'_k)_{k\in\{2,3,9\}}$ with $T'_2 = S$, $T'_3 = T$ and $T'_9 = S$, whereas equality between multisets types does not depend of the order of enumeration of its elements or the particular form with which we write the types: let us say the equality between multisets is **loose**. Thus, in system S, types and contexts are very low-level and the application typing rule can be used only in case of tight equality.

Remark. *The second use [10] that we made of System* S *is proving that every term was typable in the infinite System $\mathscr{R}$. This is not an obvious statement as for other infinitary type system, because System $\mathscr{R}$ is relevant (it forbids weakening) and there is not trivial method to type a non-normalizing term. We could not reason directly on System $\mathscr{R}$ because we needed to describe the support of a prospective derivation of a given term t and those of the types nested in the potential derivation before we decorated them. But with multiset constructions, it is impossible to see the support of a derivation as a set of pointers, whereas it is very natural with sequential constructions. This work provided a new model for pure λ-calculus.*

1.5 Reduction Choices

Regarding Subject Reduction, intersection type systems are usually not *deterministic*: if $t \to t'$ and Π is a derivation typing $t = (\lambda x.r)s$, then, the proof of the subject reduction lemma can yield several derivations Π' typing $t' = r[s/x]$. In that case, we say there are *reduction choices*.

For instance, the type system $\mathscr{R}$ is not deterministic: when we reduce t to t', there are several natural ways to produce a derivation Π'. It is possible as soon as the variable x has been assigned several times the same type σ. In sharp contrast, the use of *tight* equalities in system S makes there is only one built-in way – under the same hypotheses – to produce a derivation typing t'. Thus, system S is deterministic. We even say that the unique reduction choice is *trivial*, because, as it will turn out (§**??**), it is based upon an identity isomorphism: roughly speaking, reduction is based on the track equality *e.g.* if there is an axiom leaf typing x using track 8, then it will be substituted by an argument derivation located on track 8 and so on, even when x has been assigned several times the type S (with $S = S_8$).

The Question of Representability

Rigid types, sequence types and derivations (of system S) can be naturally collapsed into regular types, multisets types and derivations (of system $\mathscr{R}$). Actually, $\mathscr{R}$-types and multisets types are easily identifiable to equivalence classes of rigid (sequence) types. We (coinductively) **collapse** families indexed by integers into multisets. For instance, $(T_k)_{k\in K} \equiv (T'_k)_{k\in K'}$ whenever there is a bijection $\sigma : K \to K'$ (called a **track resetting**) s.t. $T'_{\sigma(k)} = T_k$. With the above examples, $(T_k)_{k\in\{2,3,8\}} \equiv (T'_k)_{k\in\{2,3,9\}}$. The equivalence relation coinductively generated by this base rule allows to see the set of $\mathscr{R}$-types (resp. multiset types) as a quotient of the set of rigid types (resp. sequence types). When a rigid type T (resp. sequence type F) is collapsed on the $\mathscr{R}$-type τ (resp. the multiset type $[\sigma_i]_{i\in I}$), we say that T (resp. F) is a **parser** of τ (resp. $[\sigma_i]_{i\in I}$).

The application rule of system $\mathscr{R}$ is based upon a *loose* equality: if, inside a rigid derivation P (of system S), we collapse every (sequence) type and apply the same recipe, we obtain a $\mathscr{R}$-derivation Π. However, it is not clear that, starting from a $\mathscr{R}$-derivation Π, we can find a rigid P that collapses into Π. For instance, it would demand that we can choose a good parser for every type introduced in an axiom rule, so that we have a (tight) equality in *all* the applications rules. Since Π can be infinite in depth or in width and the typing constraints propagate in complicated ways inside the derivation, the possibility of such a good choice is not easily granted.

Moreover, another feature of S can seem limited: in contrast to system $\mathscr{R}$, the substitutions inside an S-derivation are performed *deterministically*, while we reduce the judged term. This can been seen as restrictive, because, even when there are several occurrences of the same type, substitution can be processed only in one way in system S, which may be consider as a restriction compared to system $\mathscr{R}$. So it raises the following question: can we built a rigid representative P of an $\mathscr{R}$-derivation Π w.r.t. any reduction choice we would have done "by-hand" ? If we perform a reduction choice at each step of a reduction sequence, we speak of **reduction choices sequence**.

1.6 Contributions

We show that:

- Any *quantitative* derivation Π, approximable or not, has a rigid quantitative representative P.
- Any reduction choices sequence of length $\leqslant \omega$ can be built-in inside such a representative P, without assuming this reduction sequence to be sound (*strongly converging*, [8])

We proceed this way: we represent every quantitative $\mathscr{R}$-derivation Π by means of an *hybrid* derivation P_h (in a new type system $\mathtt{S_h}$) in which the tight equality (in the @-rule) is loosened and replaced by a congruence. Next, we endow those hybrid derivations with rigid, deterministic reduction choices (to be called *interfaces*), yielding *operable* derivations (in another type system $\mathtt{S_o}$). We show then that every

"by-hand" reduction sequence of (possible) infinite length can be encoded in an interface. The *trivial* derivations are the operable derivation (system S) in which the interface uses only identity isomorphisms. Finally, we prove that every operable derivation is *isomorphic* to a trivial derivation. This result concludes the proof of the Representation Theorem, stating that our non-idempotent, rigid intersection type system S has more expressive power than the system $\mathscr{M}$.

The most difficult point is the last one, *i.e.* establishing every $\mathscr{R}$-derivation has a *trivial* S-representative. In a finitary framework, this could be possible by studying first the derivations typing a NF (for which representation is granted [12]), and then proceeding by subject expansion. However, as we have hinted at, typability in system $\mathscr{R}$ does not imply any kind of normalization (some non-HN terms can be typed). So we will resort to *ad hoc* notions of *referent bipositions*, *syntactic polarity* and *collapsing strategy* (which is a partial reduction strategy) to reach our goal. It is to be noticed that the method we present does not rely on any kind of notion of **productive** reduction (see for instance [6]), as in other infinitary frameworks. The proofs may be found in [11].

References

[1] A. Bernadet and S. Lengrand. Non-idempotent intersection types and strong normalisation. Logical Methods in Computer Science, 9(4), 2013.

[2] A. Bucciarelli, D. Kesner, and S. R. D. Rocca. The inhabitation problem for non-idempotent intersection types. In Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings, pages 341–354, 2014.

[3] A. Bucciarelli, D. Kesner, and D. Ventura. Strong normalization through intersection types and memory. In Proc. of the 10th Int. Workshop on Logical and Semantical Frameworks, with Applications (LSFA), ENTCS, Natal, Brazil, August-September 2015.

[4] D. D. Carvalho. Sémantique de la logique linéaire et temps de calcul. PhD thesis, Université Aix-Marseille, Nov. 2007.

[5] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. Notre Dame Journal of Formal Logic, 21(4):685–693, 1980.

[6] J. Endrullis, H. H. Hansen, D. Hendriks, A. Polonsky, and A. Silva. A coinductive framework for infinitary rewriting and equational reasoning. In 26th International Conference on Rewriting Techniques and Applications, RTA 2015, June 29 to July 1, 2015, Warsaw, Poland, pages 143–159, 2015.

[7] P. Gardner. Discovering needed reductions using type theory. In Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings, pages 555–574, 1994.

[8] R. Kennaway, J. W. Klop, M. R. Sleep, and F. de Vries. Infinitary lambda calculus. Theor. Comput. Sci., 175(1):93–125, 1997.

[9] S. van Bakel. Intersection type assignment systems. Theor. Comput. Sci., 151(2):385–435, 1995.

[10] P. Vial. Coinductive intersection types are completely unsound,http://arxiv.org/abs/1612.06740, 2016.

[11] P. Vial. The collapse of the sequential intersection type system on the multiset intersection type is surjective, http://arxiv.org/abs/1610.06399, 2016.

[12] P. Vial. Infinitary intersection types as sequences: a new answer to klop's question, http://arxiv.org/abs/1610.06409, 2016.