Linear Rank Intersection Types

Fábio Reis
DCC-FCUP, University of Porto, Portugal
LIACC – Artificial Intelligence and Computer Science Laboratory, University of Porto, Portugal

Sandra Alves
DCC-FCUP, University of Porto, Portugal
LIACC – Artificial Intelligence and Computer Science Laboratory, University of Porto, Portugal
CRACS, INESC-TEC – Centre for Research in Advanced Computing Systems, Porto, Portugal

Mário Florido
DCC-FCUP, University of Porto, Portugal
LIACC – Artificial Intelligence and Computer Science Laboratory, University of Porto, Portugal

Abstract

Non-idempotent intersection types provide quantitative information about typed programs, and have been used to obtain time and space complexity measures. Intersection type systems characterize termination, so restrictions need to be made in order to make typability decidable. One such restriction consists in using a notion of finite rank for the idempotent intersection types. In this work, we define a new notion of rank for the non-idempotent intersection types. We then define a novel type system and a type inference algorithm for the λ-calculus, using the new notion of rank 2. In the second part of this work, we extend the type system and the type inference algorithm to use the quantitative properties of the non-idempotent intersection types to infer quantitative information related to resource usage.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Lambda-Calculus, Intersection Types, Quantitative Types, Tight Typings

Digital Object Identifier 10.4230/LIPIcs.TYPES.2022.8

Related Version *Full Version*: `https://arxiv.org/abs/2211.17186`

Funding *Fábio Reis*: This work was partially financially supported by Base Funding – UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory – LIACC – funded by national funds through the FCT/MCTES (PIDDAC).
Sandra Alves: Partially supported by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020.
Mário Florido: This work was partially financially supported by Base Funding – UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory – LIACC – funded by national funds through the FCT/MCTES (PIDDAC).

1 Introduction

The ability to determine upper bounds for the number of execution steps of a program in compilation time is a relevant problem, since it allows us to know in advance the computational resources needed to run the program.

Type systems are a powerful and successful tool of static program analysis that are used, for example, to detect errors in programs before running them. Quantitative type systems, besides helping on the detection of errors, can also provide quantitative information related to computational properties.

Intersection types, defined by the grammar $\sigma ::= \alpha \mid \sigma_1 \cap \cdots \cap \sigma_n \to \sigma$ (where α is a type variable and $n \geq 1$), are used in several type systems for the λ-calculus [6, 7, 18, 27] and allow λ-terms to have more than one type. Non-idempotent intersection types [16, 20, 12, 4], also known as *quantitative types*, are a flavour of intersection types in which the type constructor

$\cap$ is non-idempotent, and provide more than just qualitative information about programs. They are particularly useful in contexts where we are interested in measuring the use of resources, as they are related to the consumption of time and space in programs. Type systems based on non-idempotent intersection types, use non-idempotence to count the number of evaluation steps and the size of the result. For instance in [1], the authors define several quantitative type systems, corresponding to different evaluation strategies, for which they are able to measure the number of steps taken by that strategy to reduce a term to its normal form, and the size of the term's normal form. Typability is undecidable for intersection type systems, as it corresponds to termination. One way to get around this is to restrict intersection types to finite ranks, a notion defined by Daniel Leivant in [21] that makes typability decidable - Kfoury and Wells [19] define an intersection type system that, when restricted to any finite-rank, has principal typings and decidable type inference. Type systems that use finite-rank intersection types are still very powerful and useful. For instance, rank 2 intersection type systems [18, 26, 11] are more powerful, in the sense that they can type strictly more terms, than popular systems like the ML type system [10]. Still related to decidability of typability for finite ranks, Dudenhefner and Rehof [14] studied the problem for a notion of bounded-dimensional intersection types. This notion was previously defined in the context of type inhabitation [13], where it was used to prove decidability of type inhabitation for a non-idempotent intersection type system (the problem is known to be undecidable above rank 2, for idempotent intersection types [25]).

In this paper we present a new definition of rank for the quantitative types, which we call *linear rank* and differs from the classical one in the base case – instead of simple types, linear rank 0 intersection types are the linear types. In a non-idempotent intersection type system, every linear term is typable with a simple type (in fact, in many of those systems, only the linear terms are), which is the motivation to use linear types for the base case. The relation between non-idempotent intersection types and linearity has already been studied by Kfoury [20], de Carvalho [12], Gardner [16] and Florido and Damas [15]. Our motivation to redefine rank in the first place, has to do with our interest in using non-idempotent intersection types to estimate the number of evaluation steps of a λ-term to normal form while inferring its type, and the realization that there is a way to define rank that is more suitable for the quantitative types. We define a new intersection type system for the λ-calculus, restricted to linear rank 2 non-idempotent intersection types, and a new type inference algorithm that we prove to be sound and complete with respect to the type system.

Finally we extend our type system and inference algorithm to use the quantitative properties of the linear rank 2 non-idempotent intersection types to infer not only the type of a λ-term, but also the number of evaluation steps of the term to its normal form. The new type system is the result of a merge between our Linear Rank 2 Intersection Type System and the system for the leftmost-outermost evaluation strategy presented in [1]. The type system in [1] is a quantitative typing system extended with the notion of tight types (which provide an effective characterisation of minimal typings) that is crucial to extract exact bounds for reduction. We prove that the system gives the correct number of evaluation steps for a kind of derivation. As for the new type inference algorithm, we show that it is sound and complete with respect to the type system for the inferred types, and conjecture that the inferred measures correspond to the ones given by the type system (i.e., correspond to the number of evaluation steps of the term to its normal form, when using the leftmost-outermost evaluation strategy).

Thus, the main contributions of this paper are the following:

- A new definition of rank for non-idempotent intersection types, which we call *linear rank* (Section 3);
- A Linear Rank 2 Intersection Type System for the λ-calculus (Section 3);

- A type inference algorithm that is sound and complete with respect to the Linear Rank 2 Intersection Type System (Section 3);
- A Linear Rank 2 Quantitative Type System for the λ-calculus that derives a measure related to the number of evaluation steps for the leftmost-outermost strategy (Section 4);
- A type inference algorithm that is sound and complete with respect to the Linear Rank 2 Quantitative Type System, for the inferred types, and gives a measure that we conjecture to correspond to the number of evaluation steps of the typed term for the leftmost-outermost strategy (Section 4).

In this paper we assume that the reader is familiar with the λ-calculus [3]. From now on, in the rest of the paper, terms of the λ-calculus are considered modulo α-equivalence and we use Barendregt's variable convention [2].

2 Intersection Types

The simply typed λ-calculus is a typed version of the λ-calculus, introduced by Alonzo Church in [5] and by Haskell Curry and Robert Feys in [9]. One system that uses simple types is the Curry Type System, which was first introduced in [8] for the theory of combinators, and then modified for the λ-calculus in [9]. Typability in this system is decidable and there is an algorithm that given a term, returns its principal pair. However, the system presents some disadvantages when comparing to others, one of them being the large number of terms that cannot be typed. For example, in the Curry Type System we cannot assign a type to the λ-term $\lambda x.xx$. This term, on the other hand, can be typed in systems that use intersection types, which allow terms to have more than one type. Such a system is the Coppo-Dezani Type System [6], which was one of the first to use intersection types, and a basis for subsequent systems.

▶ **Definition 1** (Intersection types). *Intersection types $\sigma, \sigma_1, \sigma_2, \ldots \in \mathbb{T}$ are defined by the following grammar:*

$$\sigma ::= \alpha \mid \sigma_1 \cap \cdots \cap \sigma_n \to \sigma$$

where $n \geq 1$ and $\sigma_1 \cap \cdots \cap \sigma_n$ is called a sequence *of types.*

Note that intersections arise in different systems in different scopes. Here we follow several previous presentations where intersections are only allowed directly on the left-hand side of arrow types and sequences are non-empty [6, 7, 18, 27].

▶ **Notation.** *The intersection type constructor $\cap$ binds stronger than $\to$: $\alpha_1 \cap \alpha_2 \to \alpha_3$ stands for $(\alpha_1 \cap \alpha_2) \to \alpha_3$.*

▶ **Example 2.** Some examples of intersection types are:

α;
$\alpha_1 \to \alpha_2$;
$\alpha_1 \cap \alpha_2 \to \alpha_3$;
$(\alpha_1 \cap \alpha_2 \to \alpha_3) \to \alpha_4$;
$\alpha_1 \cap (\alpha_1 \to \alpha_2) \to \alpha_3$.

▶ **Definition 3** (Coppo-Dezani Type System). *In the Coppo-Dezani Type System, we say that M has type σ given the environment Γ (where the predicates of declarations are sequences), and write $\Gamma \vdash_{\mathcal{CD}} M : \sigma$, if $\Gamma \vdash_{\mathcal{CD}} M : \sigma$ can be obtained from the* derivation rules *in Figure 1, where $1 \leq i \leq n$:*

$$\Gamma \cup \{x : \sigma_1 \cap \cdots \cap \sigma_n\} \vdash_{\mathcal{CD}} x : \sigma_i \qquad \text{(Axiom)}$$

$$\frac{\Gamma \cup \{x : \sigma_1 \cap \cdots \cap \sigma_n\} \vdash_{\mathcal{CD}} M : \sigma}{\Gamma \vdash_{\mathcal{CD}} \lambda x.M : \sigma_1 \cap \cdots \cap \sigma_n \to \sigma} \qquad (\to \text{ Intro})$$

$$\frac{\Gamma \vdash_{\mathcal{CD}} M_1 : \sigma_1 \cap \cdots \cap \sigma_n \to \sigma \qquad \Gamma \vdash_{\mathcal{CD}} M_2 : \sigma_1 \ \cdots \ \Gamma \vdash_{\mathcal{CD}} M_2 : \sigma_n}{\Gamma \vdash_{\mathcal{CD}} M_1 M_2 : \sigma} \qquad (\to \text{ Elim})$$

Figure 1 Coppo-Dezani Type System.

▶ **Example 4.** For the λ-term $\lambda x.xx$ the following derivation is obtained:

$$\frac{\dfrac{\{x : \sigma_1 \cap (\sigma_1 \to \sigma_2)\} \vdash_{\mathcal{CD}} x : \sigma_1 \to \sigma_2 \qquad \{x : \sigma_1 \cap (\sigma_1 \to \sigma_2)\} \vdash_{\mathcal{CD}} x : \sigma_1}{\{x : \sigma_1 \cap (\sigma_1 \to \sigma_2)\} \vdash_{\mathcal{CD}} xx : \sigma_2}}{\vdash_{\mathcal{CD}} \lambda x.xx : \sigma_1 \cap (\sigma_1 \to \sigma_2) \to \sigma_2}$$

This system is a true extension of the Curry Type System, allowing term variables to have more than one type in the ($\to$ Intro) derivation rule and the right-hand term to also have more than one type in the ($\to$ Elim) derivation rule.

2.1 Finite Rank

Intersection type systems, like the Coppo-Dezani Type System, characterize termination, in the sense that a λ-term is strongly normalizable if and only if it is typable in an intersection type system. Thus, typability is undecidable for these systems.

To get around this, some current intersection type systems are restricted to types of finite rank [18, 26, 19, 11] using a notion of rank first defined by Daniel Leivant in [21]. This restriction makes typability decidable [19]. Despite using finite-rank intersection types, these systems are still very powerful and useful. For instance, rank 2 intersection type systems [18, 26, 11] are more powerful, in the sense that they can type strictly more terms, than popular systems like the ML type system [10].

The *rank* of an intersection type is related to the depth of the nested intersections and it can be easily determined by examining the type in tree form: a type is of rank k if no path from the root of the type to an intersection type constructor $\cap$ passes to the left of k arrows.

▶ **Example 5.** The intersection type $\alpha_1 \cap (\alpha_1 \to \alpha_2) \to \alpha_2$ (tree on the left) is a rank 2 type and $(\alpha_1 \cap \alpha_2 \to \alpha_3) \to \alpha_4$ (tree on the right) is a rank 3 type:

▶ **Definition 6** (Rank of intersection types). *Let $\mathbb{T}_0$ be the set of simple types and $\mathbb{T}_1 = \{\tau_1 \cap \cdots \cap \tau_m \mid \tau_1, \ldots, \tau_m \in \mathbb{T}_0, m \geq 1\}$ the set of sequences of simple types (written as $\vec{\tau}$). The set $\mathbb{T}_k$, of rank k intersection types (for $k \geq 2$), can be defined recursively in the following way ($n \geq 3$, $m \geq 1$):*

$$\mathbb{T}_2 = \mathbb{T}_0 \cup \{\vec{\tau} \to \sigma \mid \vec{\tau} \in \mathbb{T}_1, \sigma \in \mathbb{T}_2\}$$
$$\mathbb{T}_n = \mathbb{T}_{n-1} \cup \{\rho_1 \cap \cdots \cap \rho_m \to \sigma \mid \rho_1, \ldots, \rho_m \in \mathbb{T}_{n-1}, \sigma \in \mathbb{T}_n\}$$

▶ **Notation.** *We consider the intersection type constructor $\cap$ to be associative, commutative and non-idempotent (meaning that $\alpha \cap \alpha$ is not equivalent to α).*

We are particularly interested in non-idempotent intersection types, also known as quantitative types, because they provide more quantitative information than the idempotent ones.

3 Linear Rank Intersection Types

In the previous chapter, we mentioned several intersection type systems in which intersection is idempotent and types are rank-restricted. There are also many quantitative type systems [16, 20, 12, 4] that, on the other hand, make use of non-idempotent intersection types, for which there is no specific definition of rank.

The generalization of ranking for non-idempotent intersection types is not trivial and raises interesting questions that we will address in this chapter, along with a definition of a new non-idempotent intersection type system and a type inference algorithm.

This and the following sections cover original work that we presented at the TYPES 2022 conference [23].

3.1 Linear Rank

The set of terms typed using idempotent rank 2 intersection types and non-idempotent rank 2 intersection types is not the same. For instance, the term $(\lambda x.xx)(\lambda fx.f(fx))$ is typable with a simple type when using idempotent intersection types, but not when using non-idempotent intersection types. This comes from the two different occurrences of f in $\lambda fx.f(fx)$, which even if typed with the same type, are not contractible because intersection is non-idempotent. Note that this is strongly related to the linearity features of terms. A λ-term M is called a *linear term* if and only if, for each subterm of the form $\lambda x.N$ in M, x occurs free in N exactly once, and if each free variable of M has just one occurrence free in M. So the term $(\lambda x.xx)(\lambda fx.f(fx))$ is not typable with a non-idempotent rank 2 intersection type precisely because the term $\lambda fx.f(fx)$ is not linear.

Note that in a non-idempotent intersection type system, every linear term is typable with a simple type (in fact, in many of those systems, only the linear terms are). This motivated us to come up with a new notion of rank for non-idempotent intersection types, based on linear types (the ones derived in a linear type system – a substructural type system in which each assumption must be used exactly once, corresponding to the implicational fragment of linear logic [17]). The relation between non-idempotent intersection types and linearity was first introduced by Kfoury [20] and further explored by de Carvalho [12], who established its relation with linear logic.

Here we propose a new definition of rank for intersection types, which we call *linear rank* and differs from the classical one in the base case – instead of simple types, linear rank 0 intersection types are the linear types – and in the introduction of the functional type constructor "linear arrow" $\multimap$.

▶ **Definition 7** (Linear rank of intersection types). *Let* $\mathbb{T}_{\mathbb{L}0} = \mathbb{V} \cup \{\tau_1 \multimap \tau_2 \mid \tau_1, \tau_2 \in \mathbb{T}_{\mathbb{L}0}\}$ *be the set of* ***linear types*** *and* $\mathbb{T}_{\mathbb{L}1} = \{\tau_1 \cap \cdots \cap \tau_m \mid \tau_1, \ldots, \tau_m \in \mathbb{T}_{\mathbb{L}0}, m \geq 1\}$ *the set of sequences of linear types. The set* $\mathbb{T}_{\mathbb{L}k}$*, of* linear rank k *intersection types (for* $k \geq 2$*), can be defined recursively in the following way (*$n \geq 3$*,* $m \geq 2$*):*

$$\begin{aligned}
\mathbb{T}_{\mathbb{L}2} &= \mathbb{T}_{\mathbb{L}0} \cup \{\tau \multimap \sigma \mid \tau \in \mathbb{T}_{\mathbb{L}0}, \sigma \in \mathbb{T}_{\mathbb{L}2}\} \\
&\quad \cup \{\tau_1 \cap \cdots \cap \tau_m \to \sigma \mid \tau_1, \ldots, \tau_m \in \mathbb{T}_{\mathbb{L}0}, \sigma \in \mathbb{T}_{\mathbb{L}2}\} \\
\mathbb{T}_{\mathbb{L}n} &= \mathbb{T}_{\mathbb{L}n-1} \cup \{\rho \multimap \sigma \mid \rho \in \mathbb{T}_{\mathbb{L}n-1}, \sigma \in \mathbb{T}_{\mathbb{L}n}\} \\
&\quad \cup \{\rho_1 \cap \cdots \cap \rho_m \to \sigma \mid \rho_1, \ldots, \rho_m \in \mathbb{T}_{\mathbb{L}n-1}, \sigma \in \mathbb{T}_{\mathbb{L}n}\}
\end{aligned}$$

Initially, the idea for the change arose from our interest in using rank-restricted intersection types to estimate the number of evaluation steps of a λ-term while inferring its type. While defining the intersection type system to obtain quantitative information, we realized that the ranks could be potentially more useful for that purpose if the base case was changed to types that give more quantitative information in comparison to simple types, which is the case for linear types – for instance, if a term is typed with a linear rank 2 intersection type, one knows that each occurrence of its arguments is linear, meaning that they will be used exactly once.

The relation between the standard definition of rank and our definition of linear rank is not clear, and most likely non-trivial. Note that the set of terms typed using standard rank 2 intersection types [18, 26] and linear rank 2 intersection types is not the same. For instance, again, the term $(\lambda x.xx)(\lambda fx.f(fx))$, typable with a simple type in the standard Rank 2 Intersection Type System, is not typable in the Linear Rank 2 Intersection Type System, because, as the term $(\lambda fx.f(fx))$ is not linear and intersection is not idempotent, by Definition 7, the type of $(\lambda x.xx)(\lambda fx.f(fx))$ is now (linear) rank 3. This relation between rank and linear rank is an interesting question that will not be covered here, but one that we would like to explore in the future.

3.2 Type System

We now define a new type system for the λ-calculus with linear rank 2 non-idempotent intersection types.

▶ **Definition 8** (Substitution). *Let* $S = [N/x]$ *denote a* substitution. *Then the result of substituting the term* N *for each free occurrence of* x *in the term* M*, denoted by* $M[N/x]$ *(or* $\mathcal{S}(M)$*), is inductively defined as follows:*

$$\begin{aligned}
x[N/x] &= N; \\
x_1[N/x_2] &= x_1, \ \textit{if } x_1 \neq x_2; \\
(M_1M_2)[N/x] &= (M_1[N/x])(M_2[N/x]); \\
(\lambda x.M)[N/x] &= \lambda x.M; \\
(\lambda x_1.M)[N/x_2] &= \lambda x_1.(M[N/x_2]), \ \textit{if } x_1 \neq x_2.
\end{aligned}$$

▶ **Notation.** *We write* $M[M_1/x_1, M_2/x_2, \ldots, M_n/x_n]$ *for* $(\ldots((M[M_1/x_1])[M_2/x_2])\ldots)[M_n/x_n]$.

Composing two substitutions $\mathcal{S}_1$ and $\mathcal{S}_2$ results in a substitution $\mathcal{S}_2 \circ \mathcal{S}_1$ that when applied, has the same effect as applying $\mathcal{S}_1$ followed by $\mathcal{S}_2$.

▶ **Definition 9** (Substitution composition). *The* composition *of two substitutions* $\mathcal{S}_1 = [N_1/x_1]$ *and* $\mathcal{S}_2 = [N_2/x_2]$*, denoted by* $\mathcal{S}_2 \circ \mathcal{S}_1$*, is defined as:*

$$\mathcal{S}_2 \circ \mathcal{S}_1(M) = M[N_1/x_1, N_2/x_2].$$

Also, we consider that the operation is right-associative:

$$\mathcal{S}_1 \circ \mathcal{S}_2 \circ \cdots \circ \mathcal{S}_{n-1} \circ \mathcal{S}_n = \mathcal{S}_1 \circ (\mathcal{S}_2 \circ \cdots \circ (\mathcal{S}_{n-1} \circ \mathcal{S}_n) \dots).$$

▶ **Notation.** *From now on, we will use* α *to range over a countable infinite set* $\mathbb{V}$ *of type variables,* τ *to range over the set* $\mathbb{T}_{\mathbb{L}0}$ *of linear types,* $\vec{\tau}$ *to range over the set* $\mathbb{T}_{\mathbb{L}1}$ *of linear type sequences and* σ *to range over the set* $\mathbb{T}_{\mathbb{L}2}$ *of linear rank 2 intersection types. In all cases, we may use or not single quotes and/or number subscripts.*

▶ **Definition 10.**

- *A* statement *is an expression of the form* $M : \vec{\tau}$*, where* $\vec{\tau}$ *is called the* predicate*, and the term* M *is called the* subject *of the statement.*
- *A* declaration *is a statement where the subject is a term variable.*
- *The comma operator (,) appends a declaration to the end of a list (of declarations). The list* (Γ_1, Γ_2) *is the list that results from appending the list* Γ_2 *to the end of the list* Γ_1*.*
- *A finite list of declarations is* consistent *if and only if the term variables are all distinct.*
- *An* environment *is a consistent finite list of declarations which predicates are sequences of linear types (i.e., elements of* $\mathbb{T}_{\mathbb{L}1}$*) and we use* Γ *(possibly with single quotes and/or number subscripts) to range over environments.*
- *An environment* $\Gamma = [x_1 : \vec{\tau}_1, \dots, x_n : \vec{\tau}_n]$ *induces a partial function* Γ *with domain* $\mathsf{dom}(\Gamma) = \{x_1, \dots, x_n\}$*, and* $\Gamma(x_i) = \vec{\tau}_i$*.*
- *We write* Γ_x *for the resulting environment of eliminating the declaration of* x *from* Γ *(if there is no declaration of* x *in* Γ*, then* $\Gamma_x = \Gamma$*).*
- *We extend the notion of substitution to environments in the following way:*

$$\mathcal{S}(\Gamma) = [\mathcal{S}(x_1) : \vec{\tau}_1, \dots, \mathcal{S}(x_n) : \vec{\tau}_n] \qquad \text{if } \Gamma = [x_1 : \vec{\tau}_1, \dots, x_n : \vec{\tau}_n]$$

- *We write* $\Gamma_1 \equiv \Gamma_2$ *if the environments* Γ_1 *and* Γ_2 *are equal up to the order of the declarations.*
- *If* Γ_1 *and* Γ_2 *are environments, the environment* $\Gamma_1 + \Gamma_2$ *is defined as follows: for each* $x \in \mathsf{dom}(\Gamma_1) \cup \mathsf{dom}(\Gamma_2)$*,*

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x) & \text{if } x \notin \mathsf{dom}(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \notin \mathsf{dom}(\Gamma_1) \\ \Gamma_1(x) \cap \Gamma_2(x) & \text{otherwise} \end{cases}$$

with the declarations of the variables in $\mathsf{dom}(\Gamma_1)$ *in the beginning of the list, by the same order they appear in* Γ_1*, followed by the declarations of the variables in* $\mathsf{dom}(\Gamma_2) \setminus \mathsf{dom}(\Gamma_1)$*, by the order they appear in* Γ_2*.*

▶ **Definition 11** (Linear Rank 2 Intersection Type System). *In the Linear Rank 2 Intersection Type System, we say that* M *has type* σ *given the environment* Γ*, and write* $\Gamma \vdash_2 M : \sigma$*, if it can be obtained from the* derivation rules *in Figure 2.*

▶ **Example 12.** Let us write $\overline{\alpha}^\circ$ for the type $(\alpha \multimap \alpha)$. For the λ-term $(\lambda x.xx)(\lambda y.y)$, the following derivation is obtained:

$$\dfrac{\dfrac{\dfrac{\dfrac{[x_1 : \overline{\alpha}^\circ \multimap \overline{\alpha}^\circ] \vdash_2 x_1 : \overline{\alpha}^\circ \multimap \overline{\alpha}^\circ \qquad [x_2 : \overline{\alpha}^\circ] \vdash_2 x_2 : \overline{\alpha}^\circ}{[x_1 : \overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, x_2 : \overline{\alpha}^\circ] \vdash_2 x_1 x_2 : \overline{\alpha}^\circ}}{[x : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ) \cap \overline{\alpha}^\circ] \vdash_2 xx : \overline{\alpha}^\circ}}{[\,] \vdash_2 \lambda x.xx : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ) \cap \overline{\alpha}^\circ \to \overline{\alpha}^\circ} \qquad \dfrac{[y : \overline{\alpha}^\circ] \vdash_2 y : \overline{\alpha}^\circ}{[\,] \vdash_2 \lambda y.y : \overline{\alpha}^\circ \multimap \overline{\alpha}^\circ} \qquad \dfrac{[y : \alpha] \vdash_2 y : \alpha}{[\,] \vdash_2 \lambda y.y : \overline{\alpha}^\circ}}{[\,] \vdash_2 (\lambda x.xx)(\lambda y.y) : \overline{\alpha}^\circ}$$

$$[x : \tau] \vdash_2 x : \tau \qquad \text{(Axiom)}$$

$$\frac{\Gamma_1, x : \vec{\tau_1}, y : \vec{\tau_2}, \Gamma_2 \vdash_2 M : \sigma}{\Gamma_1, y : \vec{\tau_2}, x : \vec{\tau_1}, \Gamma_2 \vdash_2 M : \sigma} \qquad \text{(Exchange)}$$

$$\frac{\Gamma_1, x_1 : \vec{\tau_1}, x_2 : \vec{\tau_2}, \Gamma_2 \vdash_2 M : \sigma}{\Gamma_1, x : \vec{\tau_1} \cap \vec{\tau_2}, \Gamma_2 \vdash_2 M[x/x_1, x/x_2] : \sigma} \qquad \text{(Contraction)}$$

$$\frac{\Gamma, x : \tau_1 \cap \cdots \cap \tau_n \vdash_2 M : \sigma \qquad n \geq 2}{\Gamma \vdash_2 \lambda x.M : \tau_1 \cap \cdots \cap \tau_n \to \sigma} \qquad (\to \text{ Intro})$$

$$\frac{\Gamma \vdash_2 M_1 : \tau_1 \cap \cdots \cap \tau_n \to \sigma \qquad \Gamma_1 \vdash_2 M_2 : \tau_1 \ \cdots \ \Gamma_n \vdash_2 M_2 : \tau_n \qquad n \geq 2}{\Gamma, \sum_{i=1}^{n} \Gamma_i \vdash_2 M_1 M_2 : \sigma} \qquad (\to \text{ Elim})$$

$$\frac{\Gamma, x : \tau \vdash_2 M : \sigma}{\Gamma \vdash_2 \lambda x.M : \tau \multimap \sigma} \qquad (\multimap \text{ Intro})$$

$$\frac{\Gamma_1 \vdash_2 M_1 : \tau \multimap \sigma \qquad \Gamma_2 \vdash_2 M_2 : \tau}{\Gamma_1, \Gamma_2 \vdash_2 M_1 M_2 : \sigma} \qquad (\multimap \text{ Elim})$$

Figure 2 Linear Rank 2 Intersection Type System.

3.3 Type Inference Algorithm

In this section we define a new type inference algorithm for the λ-calculus (Definition 23), which is sound (Theorem 32) and complete (Theorem 35) with respect to the Linear Rank 2 Intersection Type System.

Our algorithm is based on Trevor Jim's type inference algorithm [18] for a Rank 2 Intersection Type System that was introduced by Daniel Leivant in [21], where the algorithm was briefly covered. Different versions of the algorithm were later defined by Steffen van Bakel in [26] and by Trevor Jim in [18].

Part of the definitions, properties and proofs here presented are also adapted from [18].

▶ **Definition 13** (Type substitution). *Let* $\mathbb{S} = [\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$ *denote a* type substitution, *where* $\alpha_1, \ldots, \alpha_n$ *are distinct type variables in* $\mathbb{V}$ *and* $\tau_1, \ldots, \tau_n$ *are types in* $\mathbb{T}_{\mathbb{L}_0}$.

For any τ *in* $\mathbb{T}_{\mathbb{L}_0}$, $\mathbb{S}(\tau) = \tau[\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$ *is the type obtained by simultaneously substituting* α_i *by* τ_i *in* τ, *with* $1 \leq i \leq n$.

The type $\mathbb{S}(\tau)$ *is called an* instance *of the type* τ.

The notion of type substitution can be extended to environments in the following way:

$$\mathbb{S}(\Gamma) = [x_1 : \mathbb{S}(\vec{\tau_1}), \ldots, x_n : \mathbb{S}(\vec{\tau_n})] \qquad \text{if } \Gamma = [x_1 : \vec{\tau_1}, \ldots, x_n : \vec{\tau_n}]$$

The environment $\mathbb{S}(\Gamma)$ *is called an* instance *of the environment* Γ.

If $\mathbb{S}_1 = [\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$ *and* $\mathbb{S}_2 = [\tau'_1/\alpha'_1, \ldots, \tau'_n/\alpha'_n]$ *are type substitutions such that the variables* $\alpha_1, \ldots, \alpha_n, \alpha'_1, \ldots, \alpha'_n$ *are all distinct, then the type substitution* $\mathbb{S}_1 \cup \mathbb{S}_2$ *is defined as* $\mathbb{S}_1 \cup \mathbb{S}_2 = [\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n, \tau'_1/\alpha'_1, \ldots, \tau'_n/\alpha'_n]$.

Composing two type substitutions $\mathbb{S}_1$ and $\mathbb{S}_2$ results in a type substitution $\mathbb{S}_2 \circ \mathbb{S}_1$ that when applied, has the same effect as applying $\mathbb{S}_1$ followed by $\mathbb{S}_2$.

▶ **Definition 14** (Type substitution composition). *The* composition *of two type substitutions* $\mathbb{S}_1 = [\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$ *and* $\mathbb{S}_2 = [\tau'_1/\alpha'_1, \ldots, \tau'_m/\alpha'_m]$*, denoted by* $\mathbb{S}_2 \circ \mathbb{S}_1$*, is defined as:*

$$\mathbb{S}_2 \circ \mathbb{S}_1 = [\tau'_{i_1}/\alpha'_{i_1}, \ldots, \tau'_{i_k}/\alpha'_{i_k}, \mathbb{S}_2(\tau_1)/\alpha_1, \ldots, \mathbb{S}_2(\tau_n)/\alpha_n],$$

where $\{\alpha'_{i_1}, \ldots, \alpha'_{i_k}\} = \{\alpha'_1, \ldots, \alpha'_m\} \setminus \{\alpha_1, \ldots, \alpha_n\}$.

Also, we consider that the operation is right-associative:

$$\mathbb{S}_1 \circ \mathbb{S}_2 \circ \cdots \circ \mathbb{S}_{n-1} \circ \mathbb{S}_n = \mathbb{S}_1 \circ (\mathbb{S}_2 \circ \cdots \circ (\mathbb{S}_{n-1} \circ \mathbb{S}_n) \ldots).$$

3.3.1 Unification

We now recall Robinson's unification [24], for the special case of equations involving simple types. For the unification algorithm we follow a latter (more efficient) presentation by Martelli and Montanari [22].

▶ **Definition 15** (Unification problem). *A* unification problem *is a finite set of equations* $P = \{\tau_1 = \tau'_1, \ldots, \tau_n = \tau'_n\}$*. A* unifier *(or* solution*) is a substitution* $\mathbb{S}$*, such that* $\mathbb{S}(\tau_i) = \mathbb{S}(\tau'_i)$*, for* $1 \leq i \leq n$*. We call* $\mathbb{S}(\tau_i)$ *(or* $\mathbb{S}(\tau'_i)$*) a* common instance *of* τ_i *and* τ'_i*.* P *is* unifiable *if it has at least one unifier.* $\mathcal{U}(P)$ *is the set of unifiers of* P*.*

▶ **Example 16.** The types $\alpha_1 \multimap \alpha_2 \multimap \alpha_1$ and $(\alpha_3 \multimap \alpha_3) \multimap \alpha_4$ are *unifiable*. For the type substitution $\mathbb{S} = [(\alpha_3 \multimap \alpha_3)/\alpha_1, (\alpha_2 \multimap (\alpha_3 \multimap \alpha_3))/\alpha_4]$, the *common instance* is $(\alpha_3 \multimap \alpha_3) \multimap \alpha_2 \multimap (\alpha_3 \multimap \alpha_3)$.

▶ **Definition 17** (Most general unifier). *A substitution* $\mathbb{S}$ *is a* most general unifier *(MGU) of* P *if* $\mathbb{S}$ *is the least element of* $\mathcal{U}(P)$*. That is,*

$$\mathbb{S} \in \mathcal{U}(P) \text{ and } \forall \mathbb{S}_1 \in \mathcal{U}(P). \exists \mathbb{S}_2. \mathbb{S}_1 = \mathbb{S}_2 \circ \mathbb{S}.$$

▶ **Example 18.** Consider the types $\tau_1 = (\alpha_1 \multimap \alpha_1)$ and $\tau_2 = (\alpha_2 \multimap \alpha_3)$.

The type substitution $\mathbb{S}' = [(\alpha_4 \multimap \alpha_5)/\alpha_1, (\alpha_4 \multimap \alpha_5)/\alpha_2, (\alpha_4 \multimap \alpha_5)/\alpha_3]$ is a unifier of τ_1 and τ_2, but it is not the MGU.

The MGU of τ_1 and τ_2 is $\mathbb{S} = [\alpha_3/\alpha_1, \alpha_3/\alpha_2]$. The common instance of τ_1 and τ_2 by $\mathbb{S}'$, $(\alpha_4 \multimap \alpha_5) \multimap (\alpha_4 \multimap \alpha_5)$, is an instance of $(\alpha_3 \multimap \alpha_3)$, the common instance by $\mathbb{S}$.

▶ **Definition 19** (Solved form). *A unification problem* $P = \{\alpha_1 = \tau_1, \ldots, \alpha_n = \tau_n\}$ *is in* solved form *if* $\alpha_1, \ldots, \alpha_n$ *are all pairwise distinct variables that do not occur in any of the* τ_i*. In this case, we define* $\mathbb{S}_P = [\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$.

▶ **Definition 20** (Type unification). *We define the following relation* $\Rightarrow$ *on type unification problems (for types in* $\mathbb{T}_{\mathbb{L}0}$*):*

$$\begin{array}{lcll}
\{\tau = \tau\} \cup P & \Rightarrow & P & \\
\{\tau_1 \multimap \tau_2 = \tau_3 \multimap \tau_4\} \cup P & \Rightarrow & \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \cup P & \\
\{\tau_1 \multimap \tau_2 = \alpha\} \cup P & \Rightarrow & \{\alpha = \tau_1 \multimap \tau_2\} \cup P & \\
\{\alpha = \tau\} \cup P & \Rightarrow & \{\alpha = \tau\} \cup P[\tau/\alpha] & \text{if } \alpha \in \mathsf{fv}(P) \setminus \mathsf{fv}(\tau) \\
\{\alpha = \tau\} \cup P & \Rightarrow & \mathsf{FAIL} & \text{if } \alpha \in \mathsf{fv}(\tau) \text{ and } \alpha \neq \tau
\end{array}$$

where $P[\tau/\alpha]$ corresponds to the notion of type substitution extended to type unification problems. If $P = \{\tau_1 = \tau_1', \ldots, \tau_n = \tau_n'\}$, then $P[\tau/\alpha] = \{\tau_1[\tau/\alpha] = \tau_1'[\tau/\alpha], \ldots, \tau_n[\tau/\alpha] = \tau_n'[\tau/\alpha]\}$. And $\mathsf{fv}(P)$ and $\mathsf{fv}(\tau)$ are the sets of free type variables in P and τ, respectively. Since in our system all occurrences of type variables are free, $\mathsf{fv}(P)$ and $\mathsf{fv}(\tau)$ are the sets of type variables in P and τ, respectively.

▶ **Definition 21** (Unification algorithm). *Let P be a unification problem (with types in $\mathbb{T}_{\mathbb{L}0}$). The unification function $\mathsf{UNIFY}(P)$ that decides whether P has a solution and, if so, returns the MGU of P (see [24]), is defined as:*

function UNIFY*(P)*
  while $P \Rightarrow P'$ ***do***
    $P := P'$;
  if *P is in solved form* ***then***
    return $\mathbb{S}_P$;
  else
    FAIL;

▶ **Example 22.** Consider again the types $\alpha_1 \multimap \alpha_1$ and $\alpha_2 \multimap \alpha_3$ in Example 18. For the unification problem $P = \{\alpha_1 \multimap \alpha_1 = \alpha_2 \multimap \alpha_3\}$, $\mathsf{UNIFY}(P)$ performs the following transformations over P:

$$\begin{aligned} \{\alpha_1 \multimap \alpha_1 = \alpha_2 \multimap \alpha_3\} &\Rightarrow \{\alpha_1 = \alpha_2, \alpha_1 = \alpha_3\} \cup \{\,\} &&= \{\alpha_1 = \alpha_2, \alpha_1 = \alpha_3\} \\ &\Rightarrow \{\alpha_1 = \alpha_2\} \cup \{\alpha_1 = \alpha_3\}[\alpha_2/\alpha_1] &&= \{\alpha_1 = \alpha_2, \alpha_2 = \alpha_3\} \\ &\Rightarrow \{\alpha_2 = \alpha_3\} \cup \{\alpha_1 = \alpha_2\}[\alpha_3/\alpha_2] &&= \{\alpha_1 = \alpha_3, \alpha_2 = \alpha_3\} \end{aligned}$$

and, since $\{\alpha_1 = \alpha_3, \alpha_2 = \alpha_3\}$ is in solved form, it returns the type substitution $[\alpha_3/\alpha_1, \alpha_3/\alpha_2]$.

3.3.2 Type Inference

▶ **Definition 23** (Type inference algorithm). *Let Γ be an environment, M a λ-term, σ a linear rank 2 intersection type and* UNIFY *the function in Definition 21. The function $\mathsf{T}(M) = (\Gamma, \sigma)$ defines a type inference algorithm for the λ-calculus in the Linear Rank 2 Intersection Type System, in the following way:*

1. *If $M = x$, then $\Gamma = [x : \alpha]$ and $\sigma = \alpha$, where α is a new variable;*
2. *If $M = \lambda x.M_1$ and $\mathsf{T}(M_1) = (\Gamma_1, \sigma_1)$ then:*
   a. *if $x \notin \mathsf{dom}(\Gamma_1)$, then* FAIL;
   b. *if $(x : \tau) \in \Gamma_1$, then $\mathsf{T}(M) = (\Gamma_{1x}, \tau \multimap \sigma_1)$;*
   c. *if $(x : \tau_1 \cap \cdots \cap \tau_n) \in \Gamma_1$ (with $n \geq 2$), then $\mathsf{T}(M) = (\Gamma_{1x}, \tau_1 \cap \cdots \cap \tau_n \to \sigma_1)$.*
3. *If $M = M_1 M_2$, then:*
   a. *if $\mathsf{T}(M_1) = (\Gamma_1, \alpha_1)$ and $\mathsf{T}(M_2) = (\Gamma_2, \tau_2)$,*
      then $\mathsf{T}(M) = (\mathbb{S}(\Gamma_1 + \Gamma_2), \mathbb{S}(\alpha_3))$,
      where $\mathbb{S} = \mathsf{UNIFY}(\{\alpha_1 = \alpha_2 \multimap \alpha_3, \tau_2 = \alpha_2\})$ and α_2, α_3 are new variables;
   b. *if $\mathsf{T}(M_1) = (\Gamma_1', \tau_1' \cap \cdots \cap \tau_n' \to \sigma_1')$ (with $n \geq 2$) and, for each $1 \leq i \leq n$,*
      $\mathsf{T}(M_2) = (\Gamma_i, \tau_i)$[A],
      then $\mathsf{T}(M) = (\mathbb{S}(\Gamma_1' + \sum_{i=1}^{n} \Gamma_i), \mathbb{S}(\sigma_1'))$,
      where $\mathbb{S} = \mathsf{UNIFY}(\{\tau_i = \tau_i' \mid 1 \leq i \leq n\})$;

[A] Note that Γ_i, τ_i can all be different up to renaming of variables.

c. *if* $\mathsf{T}(M_1) = (\Gamma_1, \tau \multimap \sigma_1)$ *and* $\mathsf{T}(M_2) = (\Gamma_2, \tau_2)$,
then $\mathsf{T}(M) = (\mathbb{S}(\Gamma_1 + \Gamma_2), \mathbb{S}(\sigma_1))$,
where $\mathbb{S} = \mathsf{UNIFY}(\{\tau_2 = \tau\})$;
d. *otherwise* FAIL.

▶ **Example 24.** Let us show the type inference process for the λ-term $\lambda x.xx$.

- By rule 1., $\mathsf{T}(x) = ([x : \alpha_1], \alpha_1)$.
- By rule 1., again, $\mathsf{T}(x) = ([x : \alpha_2], \alpha_2)$.
- Then by rule 3.(a), $\mathsf{T}(xx) = (\mathbb{S}([x : \alpha_1] + [x : \alpha_2]), \mathbb{S}(\alpha_4)) = (\mathbb{S}([x : \alpha_1 \cap \alpha_2]), \mathbb{S}(\alpha_4))$,
  where $\mathbb{S} = \mathsf{UNIFY}(\{\alpha_1 = \alpha_3 \multimap \alpha_4, \alpha_2 = \alpha_3\}) = [\alpha_3 \multimap \alpha_4/\alpha_1, \alpha_3/\alpha_2]$.
  So $\mathsf{T}(xx) = ([x : (\alpha_3 \multimap \alpha_4) \cap \alpha_3], \alpha_4)$.
- Finally, by rule 2.(c), $\mathsf{T}(\lambda x.xx) = ([\,], (\alpha_3 \multimap \alpha_4) \cap \alpha_3 \to \alpha_4)$.

▶ **Example 25.** Let us now show the type inference process for the λ-term $(\lambda x.xx)(\lambda y.y)$.

- From the previous example, we have $\mathsf{T}(\lambda x.xx) = ([\,], (\alpha_3 \multimap \alpha_4) \cap \alpha_3 \to \alpha_4)$.
- By rules 1. and 2.(b), for the identity, the algorithm gives $\mathsf{T}(\lambda y.y) = ([\,], \alpha_1 \multimap \alpha_1)$.
- By rules 1. and 2.(b), again, for the identity, $\mathsf{T}(\lambda y.y) = ([\,], \alpha_2 \multimap \alpha_2)$.
- Then by rule 3.(b), $\mathsf{T}((\lambda x.xx)(\lambda y.y)) = (\mathbb{S}([\,] + [\,] + [\,]), \mathbb{S}(\alpha_4)) = ([\,], \mathbb{S}(\alpha_4))$,
  where $\mathbb{S} = \mathsf{UNIFY}(\{\alpha_1 \multimap \alpha_1 = \alpha_3 \multimap \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\})$, calculated by performing the following transformations:

$$\begin{aligned}
\{\alpha_1 \multimap \alpha_1 = \alpha_3 \multimap \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\} &\Rightarrow \{\alpha_1 = \alpha_3, \alpha_1 = \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\} \\
&\Rightarrow \{\alpha_1 = \alpha_3, \alpha_3 = \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\} \\
&\Rightarrow \{\alpha_1 = \alpha_4, \alpha_3 = \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_4\} \\
&\Rightarrow \{\alpha_1 = \alpha_4, \alpha_3 = \alpha_4, \alpha_4 = \alpha_2 \multimap \alpha_2\} \\
&\Rightarrow \{\alpha_1 = \alpha_2 \multimap \alpha_2, \alpha_3 = \alpha_2 \multimap \alpha_2, \alpha_4 = \alpha_2 \multimap \alpha_2\}
\end{aligned}$$

  So $\mathbb{S} = [(\alpha_2 \multimap \alpha_2)/\alpha_1, (\alpha_2 \multimap \alpha_2)/\alpha_3, (\alpha_2 \multimap \alpha_2)/\alpha_4]$
  and $\mathsf{T}((\lambda x.xx)(\lambda y.y)) = ([\,], \alpha_2 \multimap \alpha_2)$.

Now we show several properties of our type system and type inference algorithm, in order to prove the soundness and completeness of the algorithm with respect to the system.

▶ **Notation.** *We write* $\Phi \rhd \Gamma \vdash_2 M : \sigma$ *to denote that* Φ *is a derivation tree ending with* $\Gamma \vdash_2 M : \sigma$. *In this case,* $|\Phi|$ *is the depth of the derivation tree* Φ.

▶ **Lemma 26** (Substitution). *If* $\Phi \rhd \Gamma \vdash_2 M : \sigma$, *then* $\mathbb{S}(\Gamma) \vdash_2 M : \mathbb{S}(\sigma)$ *for any substitution* $\mathbb{S}$.

▶ **Lemma 27** (Relevance). *If* $\Phi \rhd \Gamma \vdash_2 M : \sigma$, *then* $x \in \mathsf{dom}(\Gamma)$ *if and only if* $x \in \mathsf{FV}(M)$.

▶ **Lemma 28.** *If* $\mathsf{T}(M) = (\Gamma, \sigma)$, *then* $x \in \mathsf{dom}(\Gamma)$ *if and only if* $x \in \mathsf{FV}(M)$.

▶ **Corollary 29.** *From Lemma 27 and Lemma 28, it follows that if* $\mathsf{T}(M) = (\Gamma, \sigma)$ *and* $\Gamma' \vdash_2 M : \sigma'$, *then* $\mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma')$.

▶ **Lemma 30.** *If* $\Phi_1 \rhd \Gamma \vdash_2 M : \sigma$, $x \in \mathsf{FV}(M)$ *and* y *does not occur in* M, *then there exists* $\Phi_2 \rhd \Gamma[y/x] \vdash_2 M[y/x] : \sigma$, *with* $|\Phi_1| = |\Phi_2|$.

▶ **Corollary 31.** *From Lemma 30, it follows that if* $\Gamma \vdash_2 M : \sigma$, $\{x_1, \ldots, x_n\} \subseteq \mathsf{FV}(M)$ *and* $y_1, \ldots, y_n$ *are all different variables not occurring in* M, *then* $\Gamma[y_1/x_1, \ldots, y_n/x_n] \vdash_2 M[y_1/x_1, \ldots, y_n/x_n] : \sigma$.

▶ **Theorem 32** (Soundness). *If* $\mathsf{T}(M) = (\Gamma, \sigma)$*, then* $\Gamma \vdash_2 M : \sigma$*.*

▶ **Lemma 33.** *If* $\mathsf{T}(M) = (\Gamma, \sigma)$*,* $x \in \mathsf{FV}(M)$ *and* y *does not occur in* M*, then* $\mathsf{T}(M[y/x]) = (\Gamma[y/x], \sigma)$*.*

▶ **Lemma 34.** *If* $\mathsf{T}(M) = (\Gamma, \sigma)$*, with* $\Gamma \equiv (\Gamma', y_1 : \vec{\tau}_1, y_2 : \vec{\tau}_2)$*, and* y *does not occur in* M*, then* $\mathsf{T}(M[y/y_1, y/y_2]) = (\Gamma'', \sigma)$*, with* $\Gamma'' \equiv (\Gamma', y : \vec{\tau}_1 \cap \vec{\tau}_2)$*.*

▶ **Theorem 35** (Completeness). *If* $\Phi \rhd \Gamma \vdash_2 M : \sigma$*, then* $\mathsf{T}(M) = (\Gamma', \sigma')$ *(for some environment* Γ' *and type* σ'*) and there is a substitution* $\mathbb{S}$ *such that* $\mathbb{S}(\sigma') = \sigma$ *and* $\mathbb{S}(\Gamma') \equiv \Gamma$*.*

Hence, we end up with a sound and complete type inference algorithm for the Linear Rank 2 Intersection Type System.

3.4 Remarks

A λ-term M is called a λI-term if and only if, for each subterm of the form $\lambda x.N$ in M, x occurs free in N at least once. Note that our type system and type inference algorithm only type λI-terms, but we could have extended them for the *affine terms* – a λ-term M is affine if and only if, for each subterm of the form $\lambda x.N$ in M, x occurs free in N at most once, and if each free variable of M has just one occurrence free in M.

There is no unique and final way of typing affine terms. For instance, in the systems in [1], arguments that do not occur in the body of the function get the empty type $[\,]$. Since we do not allow the empty sequence in our definition and adding it would make the system more complex, we decided to only work with λI-terms.

Regarding our choice of defining environments as lists and having the rules (Exchange) and (Contraction) in the type system, instead of defining environments as sets and using the $(+)$ operation for concatenation, that decision had to do with the fact that, this way, the system is closer to a linear type system. In the Linear Rank 2 Intersection Type System, a term is linear until we need to contract variables, so using these definitions makes us have more control over linearity and non-linearity. Also, it makes the system more easily extensible for other algebraic properties of intersection. We could also have rewritten the rule $(\to$ Elim) in order not to use the $(+)$ operation, which is something we might do in the future.

The downside of choosing these definitions is that it makes the proofs (in Section 3 and Section 4) more complex, as they are not syntax directed because of the rules (Exchange) and (Contraction).

4 Resource Inference

Given the quantitative properties of the Linear Rank 2 Intersection Types, we now aim to redefine the type system and the type inference algorithm, in order to infer not only the type of a λ-term, but also parameters related to resource usage. In this case, we are interested in obtaining the number of evaluation steps of the λ-term to its normal form, for the leftmost-outermost strategy.

4.1 Type System

The new type system defined in this chapter results from an adaptation and merge between our Linear Rank 2 Intersection Type System (Definition 11) and the system for the leftmost-outermost evaluation strategy presented in [1], as that system is able to derive a measure related to the number of evaluation steps for the leftmost-outermost strategy. We then begin by adapting some definitions from [1] and others that were already introduced in Section 3.

The predicates normal and neutral defining, respectively, the leftmost-outermost normal terms and neutral terms, are in Definition 36. The predicate $\mathsf{abs}(M)$ is true if and only if M is an abstraction; $\mathsf{normal}(M)$ means that M is in normal form; and $\mathsf{neutral}(M)$ means that M is in normal form and can never behave as an abstraction, i.e., it does not create a redex when applied to an argument.

▶ **Definition 36** (Leftmost-outermost normal forms).

$$\frac{}{\mathsf{neutral}(x)} \qquad \frac{\mathsf{neutral}(M) \qquad \mathsf{normal}(N)}{\mathsf{neutral}(MN)} \qquad \frac{\mathsf{neutral}(M)}{\mathsf{normal}(M)} \qquad \frac{\mathsf{normal}(M)}{\mathsf{normal}(\lambda x.M)}$$

▶ **Definition 37** (Leftmost-outermost evaluation strategy).

$$\frac{}{(\lambda x.M)N \longrightarrow M[N/x]} \qquad \frac{M \longrightarrow M'}{\lambda x.M \longrightarrow \lambda x.M'} \qquad \frac{M \longrightarrow M' \qquad \neg\mathsf{abs}(M)}{MN \longrightarrow M'N}$$

$$\frac{\mathsf{neutral}(N) \qquad M \longrightarrow M'}{NM \longrightarrow NM'}$$

▶ **Definition 38** (Finite rank multi-types). *We define the finite rank multi-types by the following grammar:*

$$\begin{aligned}
\mathsf{tight} &::= \mathsf{Neutral} \mid \mathsf{Abs} && \text{(Tight constants)}\\
t &::= \mathsf{tight} \mid \alpha \mid t \multimap t && \text{(Rank 0 multi-types)}\\
\vec{t} &::= t \mid \vec{t} \cap \vec{t} && \text{(Rank 1 multi-types)}\\
s &::= t \mid \vec{t} \to s && \text{(Rank 2 multi-types)}
\end{aligned}$$

▶ **Definition 39.**

- *Here, a* statement *is an expression of the form $M : (\vec{\tau}, \vec{t})$, where the pair $(\vec{\tau}, \vec{t})$ is called the* predicate*, and the term M is called the* subject *of the statement.*
- *A* declaration *is a statement where the subject is a term variable.*
- *The comma operator (,) appends a declaration to the end of a list (of declarations). The list (Γ_1, Γ_2) is the list that results from appending the list Γ_2 to the end of the list Γ_1.*
- *A finite list of declarations is* consistent *if and only if the term variables are all distinct.*
- *An* environment *is a consistent finite list of declarations which predicates are pairs with a sequence from $\mathbb{T}_{\mathbb{L}1}$ as the first element and a rank 1 multi-type as the second element of the pair (i.e., the declarations are of the form $x : (\vec{\tau}, \vec{t})$), and we use Γ (possibly with single quotes and/or number subscripts) to range over environments.*
- *An environment $\Gamma = [x_1 : (\vec{\tau}_1, \vec{t}_1), \ldots, x_n : (\vec{\tau}_n, \vec{t}_n)]$ induces a partial function Γ with domain $\mathsf{dom}(\Gamma) = \{x_1, \ldots, x_n\}$, and $\Gamma(x_i) = (\vec{\tau}_i, \vec{t}_i)$.*
- *We write Γ_x for the resulting environment of eliminating the declaration of x from Γ (if there is no declaration of x in Γ, then $\Gamma_x = \Gamma$).*
- *We write $\Gamma_1 \equiv \Gamma_2$ if the environments Γ_1 and Γ_2 are equal up to the order of the declarations.*
- *If Γ_1 and Γ_2 are environments, the environment $\Gamma_1 + \Gamma_2$ is defined as follows: for each $x \in \mathsf{dom}(\Gamma_1) \cup \mathsf{dom}(\Gamma_2)$,*

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x) & \text{if } x \notin \mathsf{dom}(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \notin \mathsf{dom}(\Gamma_1) \\ (\vec{\tau}_1 \cap \vec{\tau}_2, \vec{t}_1 \cap \vec{t}_2) & \text{if } \Gamma_1(x) = (\vec{\tau}_1, \vec{t}_1) \text{ and } \Gamma_2(x) = (\vec{\tau}_2, \vec{t}_2) \end{cases}$$

with the declarations of the variables in $\mathsf{dom}(\Gamma_1)$ in the beginning of the list, by the same order they appear in Γ_1, followed by the declarations of the variables in $\mathsf{dom}(\Gamma_2) \setminus \mathsf{dom}(\Gamma_1)$, by the order they appear in Γ_2.

- *We write* $\mathsf{tight}(s)$ *if* s *is of the form* tight *and* $\mathsf{tight}(t_1 \cap \cdots \cap t_n)$ *if* $\mathsf{tight}(t_i)$ *for all* $1 \leq i \leq n$. *For* $\Gamma = [x_1 : (\vec{\tau_1}, \vec{t_1}), \ldots, x_n : (\vec{\tau_n}, \vec{t_n})]$, *we write* $\mathsf{tight}(\Gamma)$ *if* $\mathsf{tight}(\vec{t_i})$ *for all* $1 \leq i \leq n$, *in which case we also say that* Γ *is tight.*

▶ **Definition 40** (Linear Rank 2 Quantitative Type System). *In the Linear Rank 2 Quantitative Type System, we say that* M *has type* σ *and multi-type* s *given the environment* Γ, *with index* b, *and write* $\Gamma \vdash^b M : (\sigma, s)$, *if it can be obtained from the* derivation rules *in Figure 3.*

The tight rules (the t-indexed ones) are used to introduce the tight constants $\mathsf{Neutral}$ and Abs, and they are related to minimal typings. Note that the index is only incremented in rules ($\multimap$ Intro) and ($\to$ Intro), as these are used to type abstractions that will be applied, contrary to the abstractions typed with the constant Abs.

▶ **Notation.** *We write* $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$ *if* Φ *is a derivation tree ending with* $\Gamma \vdash^b M : (\sigma, s)$. *In this case,* $|\Phi|$ *is the depth of the derivation tree* Φ.

▶ **Definition 41** (Tight derivations). *A derivation* $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$ *is tight if* $\mathsf{tight}(s)$ *and* $\mathsf{tight}(\Gamma)$.

Similarly to what has been done in [1], in this section we prove that, in the Linear Rank 2 Quantitative Type System, whenever a term is tightly typable with index b, then b is exactly the number of evaluations steps to leftmost-outermost normal form.

▶ **Example 42.** Let $M = (\lambda x_1.(\lambda x_2.x_2 x_1)x_1)I$, where I is the identity function $\lambda y.y$.

Let us first consider the leftmost-outermost evaluation of M to normal form:

$$(\lambda x_1.(\lambda x_2.x_2 x_1)x_1)I \longrightarrow (\lambda x_2.x_2 I)I \longrightarrow II \longrightarrow I$$

So the evaluation sequence has length 3.

Let us write $\overline{\alpha}^\circ$ for the type $(\alpha \multimap \alpha)$ and $\overline{\mathsf{Abs}}^\circ$ for the type $\mathsf{Abs} \multimap \mathsf{Abs}$.

To make the derivation tree easier to read, let us first get the following derivation Φ for the term $\lambda x_1.(\lambda x_2.x_2 x_1)x_1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{[x_2 : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ)] \vdash^0 x_2 : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ) \qquad [x_3 : (\overline{\alpha}^\circ, \mathsf{Abs})] \vdash^0 x_3 : (\overline{\alpha}^\circ, \mathsf{Abs})}{[x_2 : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ), x_3 : (\overline{\alpha}^\circ, \mathsf{Abs})] \vdash^0 x_2 x_3 : (\overline{\alpha}^\circ, \mathsf{Abs})}}{[x_3 : (\overline{\alpha}^\circ, \mathsf{Abs})] \vdash^1 \lambda x_2.x_2 x_3 : ((\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ) \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ \multimap \mathsf{Abs})} \qquad [x_4 : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ)] \vdash^0 x_4 : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ))}{[x_3 : (\overline{\alpha}^\circ, \mathsf{Abs}), x_4 : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ)] \vdash^1 (\lambda x_2.x_2 x_3)x_4 : (\overline{\alpha}^\circ, \mathsf{Abs})}}{[x_1 : (\overline{\alpha}^\circ \cap (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ), \mathsf{Abs} \cap \overline{\mathsf{Abs}}^\circ)] \vdash^1 (\lambda x_2.x_2 x_1)x_1 : (\overline{\alpha}^\circ, \mathsf{Abs})}}{[\,] \vdash^2 \lambda x_1.(\lambda x_2.x_2 x_1)x_1 : ((\overline{\alpha}^\circ \cap (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ)) \to \overline{\alpha}^\circ, (\mathsf{Abs} \cap \overline{\mathsf{Abs}}^\circ) \to \mathsf{Abs})}$$

Then for the λ-term M, the following tight derivation is obtained:

$$\dfrac{\Phi \qquad \dfrac{[y : (\alpha, \mathsf{Neutral})] \vdash^0 y : (\alpha, \mathsf{Neutral})}{[\,] \vdash^0 I : (\overline{\alpha}^\circ, \mathsf{Abs})} \qquad \dfrac{[y : (\overline{\alpha}^\circ, \mathsf{Abs})] \vdash^0 y : (\overline{\alpha}^\circ, \mathsf{Abs})}{[\,] \vdash^1 I : (\overline{\alpha}^\circ \multimap \overline{\alpha}^\circ, \overline{\mathsf{Abs}}^\circ)}}{[\,] \vdash^3 (\lambda x_1.(\lambda x_2.x_2 x_1)x_1)I : (\overline{\alpha}^\circ, \mathsf{Abs})}$$

So indeed, the index 3 represents the number of evaluation steps to leftmost-outermost normal form.

We now show several properties of the type system, adapted from [1], in order to prove the *tight correctness* (Theorem 49).

▶ **Lemma 43** (Tight spreading on neutral terms). *If* M *is a term such that* $\mathsf{neutral}(M)$ *and* $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$ *is a typing derivation such that* $\mathsf{tight}(\Gamma)$, *then* $\mathsf{tight}(s)$.

$$[x : (\tau, t)] \vdash^0 x : (\tau, t) \qquad \text{(Axiom)}$$

$$\frac{\Gamma_1, x : (\vec{\tau_1}, \vec{t_1}), y : (\vec{\tau_2}, \vec{t_2}), \Gamma_2 \vdash^b M : (\sigma, s)}{\Gamma_1, y : (\vec{\tau_2}, \vec{t_2}), x : (\vec{\tau_1}, \vec{t_1}), \Gamma_2 \vdash^b M : (\sigma, s)} \qquad \text{(Exchange)}$$

$$\frac{\Gamma_1, x_1 : (\vec{\tau_1}, \vec{t_1}), x_2 : (\vec{\tau_2}, \vec{t_2}), \Gamma_2 \vdash^b M : (\sigma, s)}{\Gamma_1, x : (\vec{\tau_1} \cap \vec{\tau_2}, \vec{t_1} \cap \vec{t_2}), \Gamma_2 \vdash^b M[x/x_1, x/x_2] : (\sigma, s)} \qquad \text{(Contraction)}$$

$$\frac{\Gamma, x : (\tau, t) \vdash^b M : (\sigma, s)}{\Gamma \vdash^{b+1} \lambda x.M : (\tau \multimap \sigma, t \multimap s)} \qquad (\multimap \text{ Intro})$$

$$\frac{\Gamma, x : (\tau, \mathsf{tight}) \vdash^b M : (\sigma, \mathsf{tight})}{\Gamma \vdash^b \lambda x.M : (\tau \multimap \sigma, \mathsf{Abs})} \qquad (\multimap \text{ Intro}_\mathsf{t})$$

$$\frac{\Gamma, x : (\tau_1 \cap \cdots \cap \tau_n, t_1 \cap \cdots \cap t_n) \vdash^b M : (\sigma, s) \qquad n \geq 2}{\Gamma \vdash^{b+1} \lambda x.M : (\tau_1 \cap \cdots \cap \tau_n \to \sigma, t_1 \cap \cdots \cap t_n \to s)} \qquad (\to \text{ Intro})$$

$$\frac{\Gamma, x : (\tau_1 \cap \cdots \cap \tau_n, \vec{t}) \vdash^b M : (\sigma, \mathsf{tight}) \qquad \mathsf{tight}(\vec{t}) \qquad n \geq 2}{\Gamma \vdash^b \lambda x.M : (\tau_1 \cap \cdots \cap \tau_n \to \sigma, \mathsf{Abs})} \qquad (\to \text{ Intro}_\mathsf{t})$$

$$\frac{\Gamma_1 \vdash^{b_1} M_1 : (\tau \multimap \sigma, t \multimap s) \qquad \Gamma_2 \vdash^{b_2} M_2 : (\tau, t)}{\Gamma_1, \Gamma_2 \vdash^{b_1+b_2} M_1 M_2 : (\sigma, s)} \qquad (\multimap \text{ Elim})$$

$$\frac{\Gamma_1 \vdash^{b_1} M_1 : (\tau \multimap \sigma, \mathsf{Neutral}) \qquad \Gamma_2 \vdash^{b_2} M_2 : (\tau, \mathsf{tight})}{\Gamma_1, \Gamma_2 \vdash^{b_1+b_2} M_1 M_2 : (\sigma, \mathsf{Neutral})} \qquad (\multimap \text{ Elim}_\mathsf{t})$$

$$\frac{\begin{array}{c}\Gamma \vdash^b M_1 : (\tau_1 \cap \cdots \cap \tau_n \to \sigma, t_1 \cap \cdots \cap t_n \to s) \\ \Gamma_1 \vdash^{b_1} M_2 : (\tau_1, t_1) \ \cdots \ \Gamma_n \vdash^{b_n} M_2 : (\tau_n, t_n) \qquad n \geq 2\end{array}}{\Gamma, \sum_{i=1}^n \Gamma_i \vdash^{b+b_1+\cdots+b_n} M_1 M_2 : (\sigma, s)} \qquad (\to \text{ Elim})$$

$$\frac{\begin{array}{c}\Gamma \vdash^b M_1 : (\tau_1 \cap \cdots \cap \tau_n \to \sigma, \mathsf{Neutral}) \\ \Gamma_1 \vdash^{b_1} M_2 : (\tau_1, \mathsf{tight}) \ \cdots \ \Gamma_n \vdash^{b_n} M_2 : (\tau_n, \mathsf{tight}) \qquad n \geq 2\end{array}}{\Gamma, \sum_{i=1}^n \Gamma_i \vdash^{b+b_1+\cdots+b_n} M_1 M_2 : (\sigma, \mathsf{Neutral})} \qquad (\to \text{ Elim}_\mathsf{t})$$

Figure 3 Linear Rank 2 Quantitative Type System.

▶ **Lemma 44** (Properties of tight typings for normal forms). *Let M be such that* $\mathsf{normal}(M)$ *and $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$ be a typing derivation.*

(i) Tightness*: if Φ is tight, then $b = 0$.*

(ii) Neutrality*: if $s = \mathsf{Neutral}$ then* $\mathsf{neutral}(M)$.

▶ **Lemma 45** (Relevance). *If $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$, then $x \in \mathsf{dom}(\Gamma)$ if and only if $x \in \mathsf{FV}(M)$.*

▶ **Lemma 46** (Substitution and typings). *Let $\Phi \rhd \Gamma \vdash^b M_1 : (\sigma, s)$ be a derivation with $x \in \mathsf{dom}(\Gamma)$ and $\Gamma(x) = (\tau_1 \cap \cdots \cap \tau_n, t_1 \cap \cdots \cap t_n)$, for $n \geq 1$. And, for each $1 \leq i \leq n$, let $\Phi_i \rhd \Gamma_i \vdash^{b_i} M_2 : (\tau_i, t_i)$.*

Then there exists a derivation $\Phi' \rhd \Gamma_x, \sum_{i=1}^n \Gamma_i \vdash^{b+b_1+\cdots+b_n} M_1[M_2/x] : (\sigma, s)$. Moreover, if the derivations $\Phi, \Phi_1, \ldots, \Phi_n$ are tight, then so is the derivation Φ'.

Proof (Sketch). The proof is by induction on $|\Phi|$. In fact, without loss of generality, we assume that $\mathsf{FV}(M_1) \cap \mathsf{FV}(M_2) = \emptyset$, so that $\Gamma_x, \sum_{i=1}^n \Gamma_i$ is consistent. Otherwise, we could simply rename the free variables in M_1 to get M_1' (and the same derivation Φ, with the variables renamed) such that $\mathsf{FV}(M_1') \cap \mathsf{FV}(M_2) = \emptyset$. Then, our proof of the lemma considers M_1, M_2 such that $\mathsf{FV}(M_1) \cap \mathsf{FV}(M_2) = \emptyset$, obtaining a derivation Φ' (with the renamed variables) and finally we could apply the rule (Contraction) (and (Exchange), when necessary) to the variables that were renamed in M_1, in order to end up with the more general form of the derivation. ◀

We now show an important property that relates contracted terms with their linear counterpart. Basically, it says that the following diagram commutes (under the described conditions):

$$\begin{array}{ccc} M' & \xrightarrow{\beta} & N' \\ {\scriptstyle \mathcal{S}(M')}\downarrow & & \downarrow{\scriptstyle \mathcal{S}(N')} \\ M & \xrightarrow[\beta]{} & N \end{array}$$

▶ **Lemma 47.** *Let $M \longrightarrow N$ and $M = \mathcal{S}(M')$ for some substitution $\mathcal{S} = [x/x_1, x/x_2]$ where x_1, x_2 occur free in M' and x does not occur in M'. Then there exists a term N' such that $N = \mathcal{S}(N')$ and $M' \longrightarrow N'$.*

▶ **Convention 4.1.** *Without loss of generality, we assume that, in a derivation tree, all contracted variables (i.e., variables that, at some point in the derivation tree, disappear from the term and environment by an application of the (Contraction) rule) are different from any other variable in the derivation tree.*
We also assume that when applying (Contraction), the new variables that substitute the contracted ones are also different from any other variable in the derivation tree.

▶ **Lemma 48** (Quantitative subject reduction). *If $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$ is tight and $M \longrightarrow N$, then $b \geq 1$ and there exists a tight derivation Φ' such that $\Phi' \rhd \Gamma \vdash^{b-1} N : (\sigma, s)$.*

Proof (Sketch). We prove the following stronger statement:

If $M \longrightarrow N$, $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$, $\mathsf{tight}(\Gamma)$, and either $\mathsf{tight}(s)$ or $\neg\mathsf{abs}(M)$, then there exists a derivation $\Phi' \rhd \Gamma \vdash^{b-1} N : (\sigma, s)$.

The proof of this statement follows by induction on $M \longrightarrow N$. The complexity of this proof is related to the (Exchange) and (Contraction) rules. Since these rules are not syntax-directed, we cannot use M do decide which rule was last applied in the derivation, since (Exchange) and (Contraction) rules can always be the last rule applied. The proof is therefore more complex and requires the use of Convention 4.1. ◀

▶ **Theorem 49** (Tight correctness). *If $\Phi \rhd \Gamma \vdash^b M : (\sigma, s)$ is a tight derivation, then there exists N such that $M \longrightarrow^b N$ and* $\mathsf{normal}(N)$. *Moreover, if $s =$* Neutral *then* $\mathsf{neutral}(N)$.

4.2 Type Inference Algorithm

We now extend the type inference algorithm defined in Section 3 (Definition 23) to also infer the number of reduction steps of the typed term to its normal form, when using the leftmost-outermost evaluation strategy.

This is done by slightly modifying the unification algorithm in Definition 21 and the algorithm in Definition 23, which will now carry and update a measure b that relates to the number of reduction steps. First, recall Definition 20, presented in Section 3.

▶ **Definition 50** (Quantitative Unification Algorithm). *Let P be a unification problem (with types in $\mathbb{T}_{\mathbb{L}0}$). The new unification function* $\mathsf{UNIFY_Q}(P)$, *which decides whether P has a solution and, if so, returns the MGU of P and an integer b used for counting purposes in the inference algorithm, is defined as:*

function $\mathsf{UNIFY_Q}(P)$
  $b := 0$;
  while $P \Rightarrow P'$ ***do***
    if $P = \{\tau_1 \multimap \tau_2 = \tau_3 \multimap \tau_4\} \cup P_1$ ***and*** $P' = \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \cup P_1$ ***then***
      $b := b + 1$;
    $P := P'$;
  if P *is in solved form* ***then***
    return $(\mathbb{S}_P, b)$;
  else
    FAIL;

Let $\mathbb{T}_{\mathbb{L}1}$-environment be an environment as defined in Section 3, i.e., just like the definition we use in the current chapter, but the predicates are only the first element of the pair (i.e., a sequence from $\mathbb{T}_{\mathbb{L}1}$).

▶ **Definition 51** (Quantitative Type Inference Algorithm). *Let Γ be a $\mathbb{T}_{\mathbb{L}1}$-environment, M a λ-term, σ a linear rank 2 intersection type, b a quantitative measure and* $\mathsf{UNIFY_Q}$ *the function in Definition 50. The function $\mathsf{T_Q}(M) = (\Gamma, \sigma, b)$ defines a new type inference algorithm that gives a quantitative measure for the λ-calculus in the Linear Rank 2 Quantitative Type System, in the following way:*

1. *If $M = x$, then $\Gamma = [x : \alpha]$, $\sigma = \alpha$ and $b = 0$, where α is a new variable;*
2. *If $M = \lambda x.M_1$ and $\mathsf{T_Q}(M_1) = (\Gamma_1, \sigma_1, b_1)$ then:*
   a. *if $x \notin \mathsf{dom}(\Gamma_1)$, then* FAIL;
   b. *if $(x : \tau) \in \Gamma_1$, then $\mathsf{T_Q}(M) = (\Gamma_{1x}, \tau \multimap \sigma_1, b_1)$;*
   c. *if $(x : \tau_1 \cap \cdots \cap \tau_n) \in \Gamma_1$ (with $n \geq 2$), then $\mathsf{T_Q}(M) = (\Gamma_{1x}, \tau_1 \cap \cdots \cap \tau_n \to \sigma_1, b_1)$.*
3. *If $M = M_1 M_2$, then:*

a. *if* $\mathsf{T}_{\mathsf{Q}}(M_1) = (\Gamma_1, \alpha_1, b_1)$ *and* $\mathsf{T}_{\mathsf{Q}}(M_2) = (\Gamma_2, \tau_2, b_2)$,
then $\mathsf{T}_{\mathsf{Q}}(M) = (\mathbb{S}(\Gamma_1 + \Gamma_2), \mathbb{S}(\alpha_3), b_1 + b_2)$,
where $(\mathbb{S}, \_) = \mathsf{UNIFY}_{\mathsf{Q}}(\{\alpha_1 = \alpha_2 \multimap \alpha_3, \tau_2 = \alpha_2\})$ *and* α_2, α_3 *are new variables;*
b. *if* $\mathsf{T}_{\mathsf{Q}}(M_1) = (\Gamma'_1, \tau'_1 \cap \cdots \cap \tau'_n \to \sigma'_1, b_1)$ *(with* $n \geq 2$*) and, for each* $1 \leq i \leq n$, $\mathsf{T}_{\mathsf{Q}}(M_2) = (\Gamma_i, \tau_i, b_i)$,
then $\mathsf{T}_{\mathsf{Q}}(M) = (\mathbb{S}(\Gamma'_1 + \sum_{i=1}^{n} \Gamma_i), \mathbb{S}(\sigma'_1), b_1 + \sum_{i=1}^{n} b_i + b_3 + 1)$,
where $(\mathbb{S}, b_3) = \mathsf{UNIFY}_{\mathsf{Q}}(\{\tau_i = \tau'_i \mid 1 \leq i \leq n\})$*;*
c. *if* $\mathsf{T}_{\mathsf{Q}}(M_1) = (\Gamma_1, \tau \multimap \sigma_1, b_1)$ *and* $\mathsf{T}_{\mathsf{Q}}(M_2) = (\Gamma_2, \tau_2, b_2)$,
then $\mathsf{T}_{\mathsf{Q}}(M) = (\mathbb{S}(\Gamma_1 + \Gamma_2), \mathbb{S}(\sigma_1), b_1 + b_2 + b_3 + 1)$,
where $(\mathbb{S}, b_3) = \mathsf{UNIFY}_{\mathsf{Q}}(\{\tau_2 = \tau\})$*;*
d. *otherwise* FAIL.

Note that b is only increased by 1 and added the quantity given by $\mathsf{UNIFY}_{\mathsf{Q}}$ in rules 3.(b) and 3.(c), since these are the only cases in which the term M is a redex.

▶ **Example 52.** Let us show the type inference process for the λ-term $\lambda x.xx$.

- By rule 1., $\mathsf{T}_{\mathsf{Q}}(x) = ([x : \alpha_1], \alpha_1, 0)$.
- By rule 1., again, $\mathsf{T}_{\mathsf{Q}}(x) = ([x : \alpha_2], \alpha_2, 0)$.
- Then by rule 3.(a), $\mathsf{T}_{\mathsf{Q}}(xx) = (\mathbb{S}([x : \alpha_1] + [x : \alpha_2]), \mathbb{S}(\alpha_4), 0 + 0) = (\mathbb{S}([x : \alpha_1 \cap \alpha_2]), \mathbb{S}(\alpha_4), 0)$,
  where $(\mathbb{S}, \_) = \mathsf{UNIFY}_{\mathsf{Q}}(\{\alpha_1 = \alpha_3 \multimap \alpha_4, \alpha_2 = \alpha_3\}) = ([\alpha_3 \multimap \alpha_4/\alpha_1, \alpha_3/\alpha_2], 0)$.
  So $\mathsf{T}_{\mathsf{Q}}(xx) = ([x : (\alpha_3 \multimap \alpha_4) \cap \alpha_3], \alpha_4, 0)$.
- Finally, by rule 2.(c), $\mathsf{T}_{\mathsf{Q}}(\lambda x.xx) = ([\,], (\alpha_3 \multimap \alpha_4) \cap \alpha_3 \to \alpha_4, 0)$.

▶ **Example 53.** Let us now show the type inference process for the λ-term $(\lambda x.xx)(\lambda y.y)$.

- From the previous example, we have $\mathsf{T}_{\mathsf{Q}}(\lambda x.xx) = ([\,], (\alpha_3 \multimap \alpha_4) \cap \alpha_3 \to \alpha_4, 0)$.
- By rules 1. and 2.(b), for the identity, the algorithm gives $\mathsf{T}_{\mathsf{Q}}(\lambda y.y) = ([\,], \alpha_1 \multimap \alpha_1, 0)$.
- By rules 1. and 2.(b), again, for the identity, $\mathsf{T}_{\mathsf{Q}}(\lambda y.y) = ([\,], \alpha_2 \multimap \alpha_2, 0)$.
- Then by rule 3.(b), $\mathsf{T}_{\mathsf{Q}}((\lambda x.xx)(\lambda y.y)) = (\mathbb{S}([\,] + [\,] + [\,]), \mathbb{S}(\alpha_4), 0 + 0 + 0 + b_3 + 1) = ([\,], \mathbb{S}(\alpha_4), b_3 + 1)$, where $(\mathbb{S}, b_3) = \mathsf{UNIFY}_{\mathsf{Q}}(\{\alpha_1 \multimap \alpha_1 = \alpha_3 \multimap \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\})$, calculated by performing the following transformations:

$$\begin{aligned}
\{\alpha_1 \multimap \alpha_1 = \alpha_3 \multimap \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\} &\Rightarrow \{\alpha_1 = \alpha_3, \alpha_1 = \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\} \\
&\Rightarrow \{\alpha_1 = \alpha_3, \alpha_3 = \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_3\} \\
&\Rightarrow \{\alpha_1 = \alpha_4, \alpha_3 = \alpha_4, \alpha_2 \multimap \alpha_2 = \alpha_4\} \\
&\Rightarrow \{\alpha_1 = \alpha_4, \alpha_3 = \alpha_4, \alpha_4 = \alpha_2 \multimap \alpha_2\} \\
&\Rightarrow \{\alpha_1 = \alpha_2 \multimap \alpha_2, \alpha_3 = \alpha_2 \multimap \alpha_2, \alpha_4 = \alpha_2 \multimap \alpha_2\}
\end{aligned}$$

  So $\mathbb{S} = [(\alpha_2 \multimap \alpha_2)/\alpha_1, (\alpha_2 \multimap \alpha_2)/\alpha_3, (\alpha_2 \multimap \alpha_2)/\alpha_4]$
  and $b_3 = 1$ because there was performed one transformation (the first) of the form $\{\tau_1 \multimap \tau_2 = \tau_3 \multimap \tau_4\} \cup P \Rightarrow \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \cup P$.
  And then, $\mathsf{T}_{\mathsf{Q}}((\lambda x.xx)(\lambda y.y)) = ([\,], \alpha_2 \multimap \alpha_2, 1 + 1) = ([\,], \alpha_2 \multimap \alpha_2, 2)$.

Since the Quantitative Type Inference Algorithm only differs from the algorithm in Section 3 on the addition of the quantitative measure, and only infers a linear rank 2 intersection type and not a multi-type, the typing soundness (Theorem 54) and completeness (Theorem 55) are formalized in a similar way.

▶ **Theorem 54** (Typing soundness). *If* $\mathsf{T}_{\mathsf{Q}}(M) = ([x_1 : \vec{\tau}_1, \ldots, x_n : \vec{\tau}_n], \sigma, b)$*, then* $[x_1 : (\vec{\tau}_1, \vec{t}_1), \ldots, x_n : (\vec{\tau}_n, \vec{t}_n)] \vdash^{b'} M : (\sigma, s)$ *(for some measure* b' *and multi-types* $s, \vec{t}_1, \ldots, \vec{t}_n$*).*

▶ **Theorem 55** (Typing completeness). *If* $\Phi \rhd [x_1 : (\vec{\tau}_1, \vec{t}_1), \ldots, x_n : (\vec{\tau}_n, \vec{t}_n)] \vdash^b M : (\sigma, s)$, *then* $\mathsf{T}_\mathsf{Q}(M) = (\Gamma', \sigma', b')$ *(for some* $\mathbb{T}_{\mathbb{L}_1}$*-environment* Γ'*, type* σ' *and measure* b'*) and there is a substitution* $\mathbb{S}$ *such that* $\mathbb{S}(\sigma') = \sigma$ *and* $\mathbb{S}(\Gamma') \equiv [x_1 : \vec{\tau}_1, \ldots, x_n : \vec{\tau}_n]$.

As for the quantitative measure given by the algorithm, we conjecture that it corresponds to the number of evaluation steps of the typed term to normal form, when using the leftmost-outermost evaluation strategy. We strongly believe the conjecture holds, based on the attempted proofs so far and because it holds for every experimental results obtained by our implementation. We have not yet proven this property, which we formalize, in part, in the second point of the strong soundness:

▶ **Conjecture 56** (Strong soundness). *If* $\mathsf{T}_\mathsf{Q}(M) = ([x_1 : \vec{\tau}_1, \ldots, x_n : \vec{\tau}_n], \sigma, b)$, *then:*

1. *There is a derivation* $\Phi \rhd [x_1 : (\vec{\tau}_1, \vec{t}_1), \ldots, x_n : (\vec{\tau}_n, \vec{t}_n)] \vdash^{b'} M : (\sigma, s)$ *(for some measure* b' *and multi-types* $s, \vec{t}_1, \ldots, \vec{t}_n$*);*
2. *If* Φ *is a tight derivation, then* $b = b'$.

Note that the second point implies, by Theorem 49, that there exists N such that $M \longrightarrow^b N$ and $\mathsf{normal}(N)$, which is what we conjecture.

We believe that proving this conjecture is not a trivial task. A first approach could be to try to use induction on the definition of $\mathsf{T}_\mathsf{Q}(M)$. However, this does not work because the subderivations within a tight derivation are not necessarily tight. For that same reason, it is also not trivial to construct a tight derivation from the result given by the algorithm or from a non-tight derivation. Thus, in order to prove this conjecture, we believe that it will be necessary to establish a stronger relation between the algorithm and tight derivations.

5 Conclusions and Future Work

When developing a non-idempotent intersection type system capable of obtaining quantitative information about a λ-term while inferring its type, we realized that the classical notion of rank was not a proper fit for non-idempotent intersection types, and that the ranks could be quantitatively more useful if the base case was changed to types that give more quantitative information in comparison to simple types, which is the case for linear types. We then came up with a new definition of rank for intersection types based on linear types, which we call *linear rank* [23]. Based on the notion of linear rank, we defined a new intersection type system for the λ-calculus, restricted to linear rank 2 non-idempotent intersection types, and a new type inference algorithm which we proved to be sound and complete with respect to the type system.

We then merged that intersection type system with the system for the leftmost-outermost evaluation strategy presented in [1] in order to use the linear rank 2 non-idempotent intersection types to obtain quantitative information about the typed terms, and we proved that the resulting type system gives the correct number of evaluation steps for a kind of derivations. We also extended the type inference algorithm we had defined, in order to also give that measure, and showed that it is sound and complete with respect to the type system for the inferred types, and conjectured that the inferred measures correspond to the ones given by the type system.

In the future, we would like to:

- prove Conjecture 56;
- further explore the relation between our definition of linear rank and the classical definition of rank;
- extend the type systems and the type inference algorithms for the affine terms;
- adapt the Linear Rank 2 Quantitative Type System and the Quantitative Type Inference Algorithm for other evaluation strategies.

References

1 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *Proc. ACM Program. Lang.*, 2(ICFP), July 2018. doi:10.1145/3236789.

2 H. P. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science (Vol. 2): Background: Computational Structures*, pages 117–309. Oxford University Press, Inc., 1993.

3 Hendrik Pieter Barendregt. *The Lambda Calculus - Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1985.

4 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017.

5 Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940. URL: http://www.jstor.org/stable/2266170.

6 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, October 1980. doi:10.1305/ndjfl/1093883253.

7 Mario Coppo. An extended polymorphic type system for applicative languages. In Piotr Dembinski, editor, *Mathematical Foundations of Computer Science 1980 (MFCS'80), Proceedings of the 9th Symposium, Rydzyna, Poland, September 1-5, 1980*, volume 88 of *Lecture Notes in Computer Science*, pages 194–204. Springer, 1980.

8 H. B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences*, 20(11):584–590, 1934. doi:10.1073/pnas.20.11.584.

9 Haskell Brooks Curry, Robert Feys, William Craig, J Roger Hindley, and Jonathan P Seldin. *Combinatory Logic*, volume 1. North-Holland Amsterdam, 1958.

10 Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 207–212, New York, NY, USA, 1982. Association for Computing Machinery. doi:10.1145/582153.582176.

11 Ferruccio Damiani. Rank 2 intersection for recursive definitions. *Fundamenta Informaticae*, 77(4):451–488, 2007.

12 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. Phd thesis, Université Aix-Marseille II, 2007.

13 Andrej Dudenhefner and Jakob Rehof. Intersection type calculi of bounded dimension. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, pages 653–665, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009862.

14 Andrej Dudenhefner and Jakob Rehof. Typability in bounded dimension. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005127.

15 Mário Florido and Luís Damas. Linearization of the lambda-calculus and its relation with intersection type systems. *J. Funct. Program.*, 14(5):519–546, 2004. doi:10.1017/S0956796803004970.

16 Philippa Gardner. Discovering needed reductions using type theory. In *TACS*, volume 789 of *LNCS*, pages 555–574. Springer, 1994.

17 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

18 Trevor Jim. Rank 2 type systems and recursive definitions. *Massachusetts Institute of Technology, Cambridge, MA*, 1995.

19 A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 161–174. ACM, 1999.

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

21 Daniel Leivant. Polymorphic type inference. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 88–98, 1983.

22 Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4:258–282, 1982.

23 Fábio Reis, Sandra Alves, and Mário Florido. Linear rank intersection types. In *28th International Conference on Types for Proofs and Programs (TYPES 2022)*, 2022. URL: https://types22.inria.fr/files/2022/06/TYPES_2022_paper_33.pdf.

24 J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965. doi:10.1145/321250.321253.

25 Pawel Urzyczyn. The emptiness problem for intersection types. *The Journal of Symbolic Logic*, 64(3):1195–1215, 1999.

26 Steffen van Bakel. *Intersection type disciplines in lambda calculus and applicative term rewriting systems*. Phd thesis, Mathematisch Centrum, Katholieke Universiteit Nijmegen, 1993.

27 Steffen van Bakel. Rank 2 intersection type assignment in term rewriting systems. *Fundam. Informaticae*, 26(2):141–166, 1996.