Nominal Essential Intersection Types

Maurício Ayala-Rincón[a], Maribel Fernández[b], Ana Cristina Rocha-Oliveira[a], Daniel Lima Ventura[c]

[a] *Departamentos de Ciência da Computação e Matemática - Universidade de Brasília*
[b] *Department of Informatics - King's College London*
[c] *Instituto de Informática - Universidade Federal de Goiás*

Abstract

Nominal systems are an alternative approach for the treatment of variables in computational systems, where first-order syntax is generalised to provide support for the specification of binding operators. In this work, an intersection type system is presented for nominal terms. The subject reduction property is shown to hold for a specialised notion of typed nominal rewriting, thus ensuring preservation of types under computational execution.

Keywords: nominal syntax, nominal rewriting, binding, essential intersection types, subject reduction

1. Introduction

Introducing variable binders in a language that works with names requires a mechanism to deal with α-equivalence, that is, invariance of objects modulo the renaming of bound variables. In logic, the existential and universal quantifiers are examples of constructors that need the binding engine to work. For instance, it must be possible to derive the equivalence between the formulas $\exists x : x > 1$ and $\exists y : y > 1$, despite their syntactical differences. In programming languages, two programs that differ only on the choice of variable names are considered equivalent. *Nominal theories* deal with binders using atoms (or variable names) and an abstraction construct. Atom-permutations are used to deal effectively with renamings and freshness constraints. This approach was introduced by Gabbay and Pitts (1999), where the Fraenkel-Mostowski permutation model of set theory with atoms (FM-sets) is indicated as "the semantic basis of meta-logics for specifying and reasoning about formal systems involving name binding, α-conversion, etc".

Nominal syntax generalises first-order syntax by providing support for the specification of languages with binding operators. In nominal syntax, there are two kinds of variables: *atoms*, which are used to represent object-level variables and can be abstracted but not be substituted, and meta-variables, called simply *variables* or *unknowns*, which can be substituted but cannot be abstracted. Substitution of a variable by a term is closer to first-order substitution where variables act as holes to be filled by terms, possibly capturing atoms (unlike higher-order theories, where substitution is non-capturing). Moreover, β-equivalence is not a primitive notion in nominal syntax, in contrast with the higher-order and explicit substitution approaches (cf. Huet (1975); Dowek et al. (2000); Ayala-Rincón and Kamareddine (2001)). Explicit substitution calculi are associated with higher-order rewriting systems, where substitutions are manipulated explicitly; some of them use de Bruijn indices to implement the substitution operation together with α-conversion in a first-order setting (see Stehr (2000)). Using a nominal rewriting system (Fernández and Gabbay (2007)), capture-avoiding substitutions can be specified with no need to manage indices as done in some explicit substitutions calculi, since names and α-equivalence are primitive notions in nominal systems.

Type systems may help programmers to detect and avoid run-time errors in programming languages but also can be used to classify programs according to their semantics. For instance, Church's Simple Type System for the λ-calculus (see Hindley (2002)) ensures $\beta\eta$-strong normalisation, i.e. termination of computations regardless of evaluation strategies, of typable terms. However, not all strongly normalisable terms are typable in the system. For example, the term $\lambda x.x\,x$ representing the self application function is a normal form, i.e. with no computation/reduction to be performed, and yet has no type in the type system of Hindley (2002).

Intersection types were originally introduced by Coppo and Dezani-Ciancaglini (1978) as an extension of the simply typed λ-calculus, intended to characterise strong normalisation in the λ-calculus (see Barendregt et al. (2013)). A term t may have two types σ and τ in a system with intersection types, denoted by $t : \sigma \cap \tau$; thus, the term $\lambda x.x\,x$ is typeable in an intersection system, by assuming $x : (\tau \to \sigma) \cap \tau$. Indeed, Coppo et al. (1981) showed that any solvable term has a meaningful type, and Barendregt et al. (1983), provided a (filter) model for the calculus itself. Hence, the fixed

point combinator $\lambda f.(\lambda x.f\,(x\,x))\ (\lambda x.f\,(x\,x))$, which is solvable[1] although non-terminating, is also typeable in such systems. Intersection types have been successfully applied in the characterisation of termination properties beyond the λ-calculus (Bernadet and Lengrand (2013); Kesner and Ventura (2014, 2017); Piccolo (2012)). Their finitary polymorphism (as opposed to the polymorphism in System F, see Girard et al. (1989)), allows one to obtain typing systems with computationally relevant properties, such as principal typings (Wells (2002)), also known as principal pairs, and decidable typing systems (Kfoury and Wells (2004)).

Various notions of typing have been proposed for nominal theories, in order to classify terms or simply to avoid undesirable syntactic constructions. Adapting a type system proposed for λ-terms to the nominal syntax is not straightforward, since the notions of substitution are different in each system (nominal substitutions, like first-order substitutions, may capture atoms since they simply replace a variable with a term). For instance, in the simply typed λ-calculus, the well-known substitution lemma ensures that $\Gamma \vdash t[x \mapsto s] : \sigma$ holds whenever $\Gamma, x : \gamma \vdash t : \sigma$ and $\Gamma \vdash s : \gamma$ hold (see Barendregt et al. (2013), Proposition 1.2.5.). This property holds because the free variables of s are not captured. In a nominal system, one also must take into account the types assigned to atoms in the leaves of the corresponding type derivation.

This paper presents an Essential Intersection Type System for nominal terms, inspired by Bakel (1995); Bakel and Fernández (1997), which adresses the specificities of the nominal framework and provides results of preservation of typings for α-equivalent terms and subject reduction for a notion of typed nominal rewriting. Throughout Section 5, examples are given to show the necessity of the conditions added in typed matching, typed rewriting and, finally, in the theorem of subject reduction. The restrictions imposed on the nominal typed rewriting relation are inspired by the polymorphic nominal type system in Fairweather and Fernández (2018).

1.1. Related Work

Intersection types have been applied in a variety of systems, with a variety of (semantic) investigation purposes. They are used in the characterisation of strong normalisation for explicit substitution calculi in Lengrand et al.

[1] corresponding to terms with head normal form in the λ-calculus.

(2004); Bernadet and Lengrand (2013), while intersection type systems are presented for several explicit calculi with de Bruijn indices in Ventura et al. (2015), all proved to have the subject reduction property. Characterisation of termination properties were also obtained through intersection types for computational interpretations of focused sequent calculi, e.g. Ghilezan et al. (2011); Espírito Santo et al. (2012); Kesner and Ventura (2017). They are also applied in investigations of termination properties for the π-calculus (Piccolo (2012)) and the semantics of session types (Castagna et al. (2009); Padovani (2012)), when combined with union types (Dunfield (2014)).

A restriction of the type system from Barendregt et al. (1983) was introduced by Bakel (1995), called Essential Intersection Type System, while preserving its main properties. Syntax-directed systems such as the one presented in Bakel (1995) have at most one typing derivation of a given typing, as opposed to the multiplicity of derivations in the system of Barendregt et al. (1983). Bakel and Fernández (1997) presented an Essential Intersection Type System for Curryfied Term Rewrite Systems, based on the typing system in Bakel (1995). With a few restrictions on the rewrite rules, the authors were able to prove subject reduction for such systems. The system proposed in the present work is based on Bakel (1995); Bakel and Fernández (1997) with respect to the intersection type features.

On type systems for nominal syntax, Fernández and Gabbay (2006) define a rank 1 polymorphic type system that explores, for the first time, syntax-directed type inference for nominal terms. A principal type function is presented that applies to a term with a type environment and a freshness context, and returns the most general type for the given parameters. Subject reduction holds for a specialised notion of rewrite step involving types.

Fairweather (2014) follows the presentation of Fernández and Gabbay (2006) and defines simple type systems *à la* Church (where α-equivalence and freshness are redefined to take into account the typed syntax) and *à la* Curry, which are then extended to include ML-style polymorphism, and dependent types. In the latter case, an extended syntax is used, where non-capturing atom-substitution is a primitive notion. Fairweather et al. (2011) presents a preliminary version of the polymorphic system; the dependent type system is described in Fairweather et al. (2015). Typed nominal rewriting and nominal algebra, both in the Church and Curry styles, are presented in Fairweather and Fernández (2018), where conditions for subject reduction with dynamically and statically typed rewrite rules are given. The latter is a refined version of the system presented in Fairweather (2014). The system in

this work is based on Fairweather (2014); Fairweather and Fernández (2018) regarding nominal restrictions to obtain properties such as subject reduction.

Previous works such as Urban et al. (2004) present a *sort system* while defining nominal terms, but with the view to guarantee some level of well-formedness, instead of exploring the semantics provided by type systems. There, atoms are allowed to be typed only with atom sorts and only well sorted permutations are built (swappings must occur between atoms of the same sort). Pitts (2003) follows a similar approach regarding sorts, but over elements of nominal sets instead of a fixed grammar of nominal terms.

In Cheney (2009), a simple type system is presented for nominal abstract syntax, where the nominal semantics is added to the λ-calculus, with $\beta\eta$-reduction shown as a primitive notion. Using the same approach, Cheney (2012) and Pitts et al. (2015) presented dependent type systems, where a dependent name-abstraction type constructor is used in the syntax of types.

1.2. Contributions

We present an Essential Intersection Type System for nominal terms, such that typings are preserved for α-equivalent terms (see Section 4). Additionally, we present notions of typed nominal matching and typed rewriting, based on Fairweather and Fernández (2018) with appropriate modifications to deal with intersection types and the type operations introduced here. Our notion of typeability also differs from Fairweather and Fernández (2018) in that there is no restriction over permutations suspended on variables, and no condition over type derivations such as the "diamond" or "compatibility property". To ensure subject reduction the notions of rewriting and matching have to be constrained due to the capturing nature of substitutions. We show that, with this notion of rewriting, subject reduction holds for uniform rewrite systems (see Section 5).

In our approach, the Essential System for the λ-calculus introduced in Bakel (1993), is adapted to nominal terms. The syntax of types is extended with user-defined and abstraction type-constructors, and the notions of type ordering as well as the type operations of lifting, substitution and expansion are extended accordingly.

To the best of our knowledge, our type system is the first one that works with intersection types in the context of nominal terms.

1.3. Outline

Section 2 discusses nominal syntax and Section 3 presents types and corresponding operations. After that, Section 4 introduces the typing system and its basic properties and Section 5 the notions of typed matching and typed rewriting and then proves the corresponding subject reduction property. Section 6 concludes and discusses future work.

2. Nominal Syntax

We fix disjoint countably infinite collections of **atoms** (or names), **unknowns** (or variables), and **term-formers** (or function symbols). We write $\mathbb{A}$ for the set of atoms and $\mathbb{V}$ for the set of variables; $a, b, c, \ldots$ will range over distinct atoms. $X, Y, Z, \ldots$ will range over distinct unknowns. $\mathsf{f}, \mathsf{g}, \ldots$ will range over distinct term-formers. We assume that to each f is associated an **arity** $n \geq 0$. A **signature** Σ is a set of term-formers with their arities.

Definition 2.1. A **swapping** $(a\ b)$ is a bijection from $\mathbb{A}$ into $\mathbb{A}$ that exchanges a and b and fixes any other atom. **Permutations** are also bijections of the form $\pi : \mathbb{A} \to \mathbb{A}$, which change a finite number of atoms and that are represented as lists of swappings. The notation $\pi \circ \pi'$ is used for the **functional composition** of permutations, so $(\pi \circ \pi')(a) = \pi(\pi'(a))$. Then, the action of a permutation over atoms is recursively defined as:

$$id(c) = c \text{ , where } id \text{ is the null list;}$$

$$((a\ b) \circ \pi)(c) = \begin{cases} a, & \text{if } \pi(c) = b; \\ b, & \text{if } \pi(c) = a; \\ \pi(c), & \text{otherwise.} \end{cases}$$

The inverse of π is the reverse list of swappings and it is denoted by π^{-1}.

Definition 2.2. The set $\mathcal{T}(\Sigma, \mathbb{A}, \mathbb{V})$ of **(nominal) terms** is recursively defined by:

$$t \ ::= \ a \mid \pi \cdot X \mid [a]t \mid \mathsf{f}(t_1, \ldots, t_n) \text{ where } n \text{ is the arity of } \mathsf{f}.$$

Call $\pi \cdot X$ a **suspension** and $[a]t$ an **(atom-)abstraction**; it represents '$x.e$' or '$x.\phi$' in expressions like '$\lambda x.e$' or '$\forall x.\phi$'.

Actions of permutations can be homomorphically extended over terms. This means that permutations only change atoms and are accumulated into suspensions. A precise definition is given below.

Definition 2.3. Define $\pi \bullet t$, called a **permutation action**, by:

$$\begin{array}{ll} \pi \bullet a = \pi(a) & \pi \bullet (\pi' \cdot X) = (\pi \circ \pi') \cdot X \\ \pi \bullet [a]t = [\pi(a)](\pi \bullet t) & \pi \bullet \mathsf{f}(t_1, \ldots, t_n) = \mathsf{f}(\pi \bullet t_1, \ldots, \pi \bullet t_n) \end{array}$$

Example 2.4. Let $\prod$ and $+$ (with infix notation) be ternary and binary symbols of a signature Σ, respectively. Consider $\prod_{i=Y}^{Z} X$ the syntactic sugar of $\prod([i]X, Y, Z)$ and the permutation $(m\,k) \circ (k\,n)$ with its inverse $(k\,n) \circ (m\,k)$. One can observe the action of both permutations over the term $\prod_{i=m}^{k}(i + X)$:

$$(m\,k) \circ (k\,n) \bullet \prod_{k=m}^{n}(k + X) = \prod_{n=k}^{m}(n + (m\,k) \circ (k\,n) \cdot X)$$

$$(k\,n) \circ (m\,k) \bullet \prod_{k=m}^{n}(k + X) = \prod_{m=n}^{k}(m + (k\,n) \circ (m\,k) \cdot X).$$

One important observation is that the variables in suspensions work as meta-variables, where a substitution that replaces unknowns by terms is a primitive notion. With this in mind, it is reasonable that nominal variables are not 'abstractable'. The denomination 'suspension' for $\pi \cdot X$ has to do with the fact that the permutation π cannot indeed be applied to X until the instance of this variable is known; so it is suspended.

Next, the meta-action of a permutation on terms is defined. This kind of application of permutations is useful when working with rewriting, as in Section 5, since we can always rename all atoms in a rule before performing a rewrite step.

Definition 2.5. Define ${}^{\pi}t$ the **meta-action** of π on t by:

$$\begin{array}{ll} {}^{\pi}a = \pi(a) & {}^{\pi}(\pi' \cdot X) = {}^{\pi}\pi' \cdot X \\ {}^{\pi}([a]t) = [\pi(a)]{}^{\pi}t & {}^{\pi}\mathsf{f}(t_1, \ldots, t_n) = \mathsf{f}({}^{\pi}t_1, \ldots, {}^{\pi}t_n), \end{array}$$

where ${}^{\pi}id = id$ and ${}^{\pi}((a\ b) \circ \pi') = (\pi(a)\ \pi(b)) \circ {}^{\pi}\pi'$.

The meta-action of permutations affects only atoms in terms (it does not suspend on variables, in contrast to the permutation action of Definition 2.3).

Example 2.6. Consider the same signature of Example 2.4. Notice that the permutations do not suspend on unknowns when applying the meta-action of them.

$${}^{(m\,k)\circ(k\,n)}\prod_{k=m}^{n}(k + X) = \prod_{n=k}^{m}(n + X) \qquad {}^{(k\,n)\circ(m\,k)}\prod_{k=m}^{n}(k + X) = \prod_{m=n}^{k}(m + X).$$

Definition 2.7. A **substitution on unknowns**, ranged over $\theta, \vartheta, \mu, \ldots$, is a function from unknowns to terms with finite domain (that is, only a finite set of unknowns are mapped to terms). Each substitution θ is represented as a list of **nuclear substitutions**, which are pairs of the form $[X \mapsto s]$, and their **action** over terms is defined as:

$$\begin{array}{rl} a[X \mapsto s] = a & (\pi \cdot X)[X \mapsto s] = \pi \bullet s \\ ([a]t)[X \mapsto s] = [a](t[X \mapsto s]) & (\pi \cdot Y)[X \mapsto s] = \pi \cdot Y \\ \mathsf{f}(t_1, \ldots, t_n)[X \mapsto s] = \mathsf{f}(t_1[X \mapsto s], \ldots, t_n[X \mapsto s]) & \end{array}$$

We write id for the substitution when the set of changed unknowns is empty (it will always be clear whether we mean 'id the identity substitution' or 'id the identity permutation'). The juxtaposition of substitutions $\theta\theta'$ denotes the composition of the respective functions, mapping each X into $(X\theta)\theta'$. So, the action of θ over terms is defined inductively by:

$$t\, id = t \qquad\qquad t\,(\theta[X \mapsto s]) = (t\,\theta)[X \mapsto s].$$

Remark 2.8. This notion of substitution is different from the simultaneous application of nuclear substitutions; instead, we apply the nuclear substitutions one by one and permit non-idempotent substitutions. This approach is closer to triangular substitutions as explored in Kumar and Norrish (2010).

Definition 2.9. A **freshness (constraint)** is a pair $a\#t$ of an atom a and a term t. We call a freshness of the form $a\#X$ **primitive**, and a finite set of primitive freshnesses a **freshness context**. Δ and ∇ will range over freshness contexts.

We denote by $\nabla\theta$ the set $\{a\#\theta(X) |\ a\#X \in \nabla\}$ of freshness constraints. The meta-action of permutations can also be extended to contexts as ${}^{\pi}\nabla = \{\pi(a)\#X |\ a\#X \in \nabla\}$.

A **freshness judgement** is a pair $\Delta \vdash a\#t$ of a freshness context and a freshness constraint. An **α-equivalence judgement** is a tuple $\Delta \vdash s \approx_\alpha t$ of a freshness context and two terms. The **derivable** freshness and α-equivalence judgements are obtained by the rules in Table 1, where $ds(\pi, \pi') = \{a \in \mathbb{A} \mid \pi(a) \neq \pi'(a)\}$. For A a finite set of atoms, $A\#X$ denotes the freshness context $\{a\#X \mid a \in A\}$. We call $ds(\pi, \pi')$ the **difference set** of permutations π and π'.

Table 1: Freshness and α-equality

$$\frac{}{\nabla \vdash a\#b}\ (\#\mathbf{ab}) \qquad \frac{\pi^{-1}(a)\#X \in \nabla}{\nabla \vdash a\#\pi\cdot X}\ (\#\mathbf{X}) \qquad \frac{}{\nabla \vdash a\#[a]s}\ (\#[\mathbf{a}])$$

$$\frac{\nabla \vdash a\#s_1 \quad \ldots \quad \nabla \vdash a\#s_n}{\nabla \vdash a\#\mathsf{f}(s_1,\ldots,s_n)}\ (\#\mathsf{f}) \qquad \frac{\nabla \vdash a\#s}{\nabla \vdash a\#[b]s}\ (\#[\mathbf{b}])$$

$$\frac{}{\nabla \vdash a \approx_\alpha a}\ (\approx_\alpha \mathbf{a}) \qquad \frac{ds(\pi,\pi')\#X \subseteq \nabla}{\nabla \vdash \pi\cdot X \approx_\alpha \pi'\cdot X}\ (\approx_\alpha \mathbf{X})$$

$$\frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha[\mathbf{a}]) \qquad \frac{\nabla \vdash s \approx_\alpha (a\ b)\bullet t \quad \nabla \vdash a\#t}{\nabla \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha[\mathbf{b}])$$

$$\frac{\nabla \vdash s_1 \approx_\alpha t_1 \quad \ldots \quad \nabla \vdash s_n \approx_\alpha t_n}{\nabla \vdash \mathsf{f}(s_1,\ldots,s_n) \approx_\alpha \mathsf{f}(t_1,\ldots,t_n)}\ (\approx_\alpha \mathsf{f})$$

Definition 2.10. The set $Pos(t)$ of **positions** of a term t is defined below. Note that ϵ is the only position in atoms and suspensions.

$$\frac{}{\epsilon \in Pos(t)}\ (\mathbf{p}_\epsilon) \qquad \frac{p \in Pos(t)}{1\cdot p \in Pos([a]t)}\ (\mathbf{p}_{[\mathbf{a}]})$$

$$\frac{p \in Pos(t_i) \quad (1 \le i \le n)}{i\cdot p \in Pos(\mathsf{f}(t_1,\ldots,t_i,\ldots,t_n))}\ (\mathbf{p}_\mathsf{f})$$

The notation $t|_p$ represents the **subterm** of t **at position** p, which is defined by:

$$t|_\epsilon = t \qquad [a]t|_{1\cdot p} = t|_p \qquad \mathsf{f}(t_1,\ldots,t_i,\ldots,t_n)|_{i\cdot p} = t_i|_p \quad (1 \le i \le n)$$

If $p \in Pos(s)$, then $s[p{\leftarrow}t]$ denotes the replacement of $s|_p$ at position p by t in s.

Definition 2.11. The function *atms* is used to compute the atoms in permutations and in terms. The set $atms(\pi)$ is defined by:

$$atms(id) \;=\; \varnothing \qquad atms((a\ b)\circ\pi) \;=\; \{a,b\} \cup atms(\pi).$$

The notations $atms(t)$ and $unkn(t)$ will be used to represent the set of atoms and unknowns in a term t, respectively. They are defined by:

$$\begin{array}{rlrl} atms(a) &= \{a\} & atms(\pi \cdot X) &= atms(\pi) \\ atms([a]t) &= atms(t) \cup \{a\} & atms(\mathtt{f}(t_1, \ldots, t_n)) &= \bigcup_i atms(t_i) \\ unkn(a) &= \varnothing & unkn(\pi \cdot X) &= \{X\} \\ unkn([a]t) &= unkn(t) & unkn(\mathtt{f}(t_1, \ldots, t_n)) &= \bigcup_i unkn(t_i) \end{array}$$

3. Types, ordering and operations

The next definition introduces the set of types considered in this paper.

Definition 3.1. Let $\mathcal{C}$ be a set of type constructors, each one with a fixed arity, and let $\mathcal{V}$ be a countably infinite set of type variables, respectively. The set $\mathcal{T}_s$ is defined containing the **strict types**, which are generated by:

$$\tau ::= \varphi \mid \tau_1 \cap \cdots \cap \tau_k \to \tau \mid [\tau_1 \cap \cdots \cap \tau_k]\tau \mid \mathtt{C}(\tau_1, \ldots, \tau_n),$$

where $\varphi \in \mathcal{V}$, $\mathtt{C} \in \mathcal{C}$ and n is the arity of $\mathtt{C}$ and k might be possibly 0. Define $\mathcal{T}_S$ as the set of **intersection types**, which are built using types in $\mathcal{T}_s$ as

$$\sigma ::= \tau_1 \cap \cdots \cap \tau_k, \;\; k \geq 0.$$

The symbol ω represents the intersection of zero strict types.

Notation. Let $\bigcap_{i=1}^{n} \tau_i$ denote a type in $\mathcal{T}_S$ whenever $\tau_i \in \mathcal{T}_s$ thus for $n = 1$ in particular $\bigcap_{i=1}^{n} \tau_i \in \mathcal{T}_s$. The set $Vars(\sigma)$ contains each $\varphi \in \mathcal{V}$ that occurs in σ.

We now extend the signature of terms (given in Definition 2.2) as follows: each function symbol $\mathtt{f} \in \Sigma$ is accompanied by its arity $n \geq 0$ and a **type declaration** denoted by $\Sigma_{\mathtt{f}}$, which is a type in $\mathcal{T}_S$.

Notice that the grammar of types (given in Definition 3.1) mixes abstraction types from Fernández and Gabbay (2006) with intersection types in the style of Bakel and Fernández (1997). In essence, these abstraction types behave as arrow types in the sense that contravariance in the "domain" is supposed when comparing arrow types as well as abstraction types (cf. Table 2) and the type in the right-hand side cannot be an intersection.

Example 3.2. Let Π be a ternary symbol of a signature Σ and $[\mathtt{nat}]\varphi \to \mathtt{nat} \to \mathtt{nat} \to \varphi$ be the type declaration of Π. Consider $\prod_{i=n}^{k} X$ the syntactic sugar of $\prod([i]X, n, k)$. This type declaration requires the first argument of Π to be an abstraction type and the second and third ones to be natural numbers; the type of the resulting term is φ.

Definition 3.3. A **type environment** Γ is a finite set of **type annotations** of the form $X : \sigma$ or $a : \sigma$, where each X/a appears only in one annotation. The type annotations of X and a in Γ are denoted by Γ_X and Γ_a, respectively.

The connective $\bowtie$ represents an **updating** in an environment Γ. In this way, if a and X are not annotated in Γ, then $\Gamma \bowtie a : \sigma$ and $\Gamma \bowtie X : \sigma$ denote $\Gamma \cup \{a : \sigma\}$ and $\Gamma \cup \{X : \sigma\}$, respectively. Otherwise, they respectively denote $\Gamma \setminus \Gamma_a \cup \{a : \sigma\}$ and $\Gamma \setminus \Gamma_X \cup \{X : \sigma\}$.

The meta-action of permutation (Definition 2.5) can be extended to environments: ${}^{\pi}\Gamma = \{\pi(a) : \Gamma_a \mid a \in \Gamma\} \cup \{X : \Gamma_X \mid X \in \Gamma\}$.

Pairs $\langle \Gamma, \sigma \rangle$ of an environment Γ and a type σ will be called *typings* with respect to a term, whose definition will be introduced formally in Definition 4.1. In Table 2, a partial order is defined for types, environments and this kind of pairs.

Table 2: Relation $\leq$ between types

$$(\leq_{\cap_{\mathbf{E}}}) \ \tau_1 \cap \cdots \cap \tau_n \leq \tau_i \ , \forall 1 \leq i \leq n \qquad (\leq_{\mathbf{Trans}}) \ \frac{\sigma \leq \tau \qquad \tau \leq \rho}{\sigma \leq \rho}$$

$$(\leq_{\mathbf{abs}}) \ \frac{\sigma \leq \gamma \qquad \rho \leq \tau}{[\gamma]\rho \leq [\sigma]\tau} \qquad (\leq_{\rightarrow}) \ \frac{\sigma \leq \gamma \qquad \rho \leq \tau}{\gamma \rightarrow \rho \leq \sigma \rightarrow \tau}$$

$$(\leq_{\cap_{\mathbf{I}}}) \ \frac{\sigma \leq \tau_1 \qquad \ldots \qquad \sigma \leq \tau_n}{\sigma \leq \tau_1 \cap \cdots \cap \tau_n}$$

$$(\leq_{\mathbf{\Gamma}}) \ \frac{\forall y \in \Gamma : (y \in \Phi) \wedge (\Phi_y \leq \Gamma_y) \qquad y \in \mathbb{A} \cup \mathbb{V}}{\Phi \leq \Gamma}$$

$$(\leq_{\mathbf{pair}}) \ \frac{\Gamma' \leq \Gamma \qquad \sigma \leq \sigma'}{\langle \Gamma, \sigma \rangle \leq \langle \Gamma', \sigma' \rangle}$$

The lemma that follows is taken from Kamareddine and Nour (2007), where the difference between our types and theirs is the presence of abstraction types here. Since, the ordering $\leq$ does not mix abstraction and arrow types in rules of Table 2 and both are contravariant in the domain, the verification that the properties for "$\_ \rightarrow \_$" hold for "$[\_]\_$" is straightforward. These properties will be used in the next section.

Lemma 3.4 (≤-Inversion, Kamareddine and Nour (2007))**.** *The relation* ≤ *between types satisfies the following properties:*

1. $\gamma \leq \sigma$ *implies that* $\gamma = \bigcap_{i=1}^{n} \tau_i$ *and* $\sigma = \bigcap_{j=1}^{m} \tau'_j$ *and, for all* $1 \leq j \leq m$, *there exists* $1 \leq i \leq n$ *such that* $\tau_i \leq \tau'_j$.

2. $[\gamma]\tau \leq \sigma$ *implies that* $\sigma = \bigcap_{i=1}^{n} [\gamma_i]\rho_i$, *where* $\gamma_i \leq \gamma$ *and* $\tau \leq \rho_i$ *for all* $1 \leq i \leq n$.

Now, the type operations of *lifting*, *(type)* substitution and *expansion* are introduced. While type substitution and weakening are the operations associated with principal pairs in a simple typing system, they are not enough in a system with intersection types. A type substitution replacing type variables by intersection types is unsound, since intersection types may appear on the right of an →. Type substitutions are then defined to replace type variables by types or ω, the latter in a sensible way. Expansion is the operation introducing intersections in a typing, compatible with typing derivations. Finally, the lifting operation is related to ≤. Combining those three operations allows one to obtain derivable typings for a term t, from a pair corresponding to a typing derivation for t.

Definition 3.5. An operation of **lifting** $L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle} : \mathcal{T}_S \to \mathcal{T}_S$ is defined by a pair of pairs $\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle$ such that $\langle\Gamma,\sigma\rangle \leq \langle\Gamma',\sigma'\rangle$, following the rules:

- $L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\sigma) = \sigma'$,
- $L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\gamma) = \gamma$, if $\sigma \neq \gamma$.

Liftings extend naturally to type environments and to typings:

- $L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\Phi) = \Gamma' \bowtie (\Phi \setminus \Gamma)$
- $L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\langle\Phi,\gamma\rangle) = \langle(L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\Phi)), (L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\gamma))\rangle$.

  Observe that, if $\Phi = \Gamma$, then $L_{\langle\langle\Gamma,\sigma\rangle,\langle\Gamma',\sigma'\rangle\rangle}(\Gamma) = \Gamma'$.

Example 3.6. Consider $L = L_{\langle\langle\{X:\varphi_1\cap\varphi_2\},\varphi_1\cap\varphi_2\rangle,\langle\{X:\varphi_1\cap\varphi_2\cap\varphi_3\},\varphi_2\rangle\rangle}$. So, one can obtain a lifted pair as follows on the pair below:

$$L(\langle\{X : \varphi_1 \cap \varphi_2\}, \varphi_1 \cap \varphi_2\rangle) = \langle\{X : \varphi_1 \cap \varphi_2 \cap \varphi_3\}, \varphi_2\rangle$$

However, if the lifting is applied to a different environment, then the result may vary:

$$L(\{X : \varphi_1 \cap \varphi_2, a : \rho\}) = \{X : \varphi_1 \cap \varphi_2 \cap \varphi_3, a : \rho\},$$

$$L(\varnothing) = \{X : \varphi_1 \cap \varphi_2 \cap \varphi_3\},$$

$$L(\{X : \varphi_1 \cap \varphi_5\}) = \{X : \varphi_1 \cap \varphi_5\}.$$

In any case, we have an environment smaller than the original with respect to $\leq$.

Definition 3.7. A **type substitution** $(\varphi \mapsto \alpha) : \mathcal{T}_S \to \mathcal{T}_S$, where φ is a type variable and $\alpha \in \mathcal{T}_s \cup \{\omega\}$, is defined as follows (note that substitutions also extend to type environments and typings in the natural way):

1. $(\varphi \mapsto \alpha)(\varphi) = \alpha$,
2. $(\varphi \mapsto \alpha)(\varphi') = \varphi'$, if $\varphi \neq \varphi'$,
3. $(\varphi \mapsto \alpha)(\mathtt{C}(\tau_1, \ldots, \tau_n)) = \omega$, if $(\varphi \mapsto \alpha)(\tau_i) = \omega$, for some $i = 1, \ldots, n$,
4. $(\varphi \mapsto \alpha)(\mathtt{C}(\tau_1, \ldots, \tau_n)) = \mathtt{C}((\varphi \mapsto \alpha)(\tau_1), \ldots, (\varphi \mapsto \alpha)(\tau_n))$, if $(\varphi \mapsto \alpha)(\tau_i) \neq \omega$, for all $i = 1, \ldots, n$,
5. $(\varphi \mapsto \alpha)(\sigma \to \tau) = \omega$, if $(\varphi \mapsto \alpha)(\tau) = \omega$,
6. $(\varphi \mapsto \alpha)(\sigma \to \tau) = (\varphi \mapsto \alpha)(\sigma) \to (\varphi \mapsto \alpha)(\tau)$, if $(\varphi \mapsto \alpha)(\tau) \neq \omega$,
7. $(\varphi \mapsto \alpha)([\sigma]\tau) = \omega$, if $(\varphi \mapsto \alpha)(\tau) = \omega$,
8. $(\varphi \mapsto \alpha)([\sigma]\tau) = [(\varphi \mapsto \alpha)(\sigma)](\varphi \mapsto \alpha)(\tau)$, if $(\varphi \mapsto \alpha)(\tau) \neq \omega$,
9. $(\varphi \mapsto \alpha)(\tau_1 \cap \cdots \cap \tau_n) = (\varphi \mapsto \alpha)(\rho_1) \cap \cdots \cap (\varphi \mapsto \alpha)(\rho_m)$, where $\{\rho_1, \ldots, \rho_m\} = \{\tau_i \in \{\tau_1, \ldots, \tau_n\} \mid (\varphi \mapsto \alpha)(\tau_i) \neq \omega\}$,
10. $(\varphi \mapsto \alpha)(\Gamma) = \{y : (\varphi \mapsto \alpha)(\gamma) \mid y : \gamma \in \Gamma\}$,
11. $(\varphi \mapsto \alpha)(\langle \Gamma, \gamma \rangle) = \langle (\varphi \mapsto \alpha)(\Gamma), (\varphi \mapsto \alpha)(\gamma) \rangle$.

We may say substitution instead of type substitution for the sake of simplicity, whenever clear from the context.

Example 3.8. Take a type substitution $S = (\varphi_1 \mapsto \mathtt{nat})$. So, we can instantiate the following pair as follows:

$$S(\langle\{X : [\varphi_1]\varphi_2\}, \varphi_1\rangle) = \langle\{X : [\mathtt{nat}]\varphi_2\}, \mathtt{nat}\rangle.$$

For $S' = (\varphi_2 \mapsto \omega) \circ S$, we must be careful:

$$S'(\varphi_1 \cap \varphi_2) = \mathtt{nat},$$

$$S'(\varphi_1 \to \varphi_2) = \omega.$$

The next lemma asserts the compatibility of type substitutions with the relation $\leq$.

Lemma 3.9 (Compatibility of substitutions with $\leq$)**.** *Let S be a type substitution.*

1. *$\sigma \leq \gamma$ implies $S(\sigma) \leq S(\gamma)$.*
2. *$\Phi \leq \Gamma$ implies $S(\Phi) \leq S(\Gamma)$.*

Proof. 1. The proof is by induction on the derivation of $\sigma \leq \gamma$. The interesting cases are when the rules $(\leq_{\mathbf{abs}})$ or $(\leq_{\to})$ are the last used ones. Since those cases are very similar, only the case for $(\leq_{\to})$ will be presented here.

We have $\sigma = \sigma' \to \rho$ and $\gamma = \gamma' \to \tau$. As premises of the rule, it holds that $\gamma' \leq \sigma'$ and $\rho \leq \tau$. By IH, $S(\gamma') \leq S(\sigma')$ and $S(\rho) \leq S(\tau)$. If $S(\rho) = \omega$, then $S(\tau) = \omega$ by Lemma 3.4 part 1. In this case, $S(\sigma' \to \rho) = \omega \leq \omega = S(\gamma' \to \tau)$. Otherwise, $S(\sigma' \to \rho) = S(\sigma') \to S(\rho) \leq S(\gamma') \to S(\tau) = S(\gamma' \to \tau)$ using rule $(\leq_{\to})$.

2. It follows by part 1 and rule $(\leq_{\mathbf{\Gamma}})$.

□

The next two definitions are adapted from Bakel (2011), to deal with abstraction types and types with constructors. In particular, the notion of last type variable of a type, which is used to ensure that the operation of expansion produces a valid type, is generalised to a set of type variables since type constructors may have arity $n > 1$.

Definition 3.10. The set of **last type variables** of a type τ in $\mathcal{T}_s$ is denoted by $LV(\tau)$ and defined by:

1. $LV(\varphi)$ is $\{\varphi\}$,
2. $LV(\mathtt{C}(\tau_1, \ldots, \tau_n))$ is $\bigcup_{i=1}^{n} LV(\tau_i)$,
3. $LV(\sigma \to \tau)$ and $LV([\sigma]\tau)$ is $LV(\tau)$.

In the operation of type expansion, we must ensure that intersection types are not introduced where strict types forbid them (i.e., in the right-hand side of arrow and abstraction types, and in arguments of constructor types; see Definition 3.1). The notion of last type variable is used to analyse cases in the definition of expansion below.

Definition 3.11. For all $\psi \in \mathcal{T}_s$, $k \geq 2$, environment Γ and type σ, the quadruple $\langle \psi, k, \Gamma, \sigma \rangle$ determines an **expansion** $Exp_{\langle \psi,k,\Gamma,\sigma \rangle} : \mathcal{T}_S \to \mathcal{T}_S$, which proceeds as follows:

1. The set of type variables $\mathcal{V}_\psi(\Gamma, \sigma)$ affected by $Exp_{\langle \psi,k,\Gamma,\sigma \rangle}$ is built by:
   a) If φ occurs in ψ, then $\varphi \in \mathcal{V}_\psi(\Gamma, \sigma)$,
   b) If $\tau \in \mathcal{T}_s$ is a subtype of Γ or σ and $LV(\tau) \cap \mathcal{V}_\psi(\Gamma, \sigma) \neq \varnothing$, then all type variables of τ are in $\mathcal{V}_\psi(\Gamma, \sigma)$.
2. Suppose $\mathcal{V}_\psi(\Gamma, \sigma) = \{\varphi_1, \ldots, \varphi_m\}$. Take $m \times k$ different type variables $\varphi_1^1, \ldots, \varphi_1^k, \ldots, \varphi_m^1, \ldots, \varphi_m^k$, disjoint from the variables in Γ and σ. For each $i = 1, \ldots, k$, consider a type substitution S_i such that $S_i(\varphi_j) = \varphi_j^i$, for all $j = 1, \ldots, m$.
3. $Exp_{\langle \psi,k,\Gamma,\sigma \rangle}(\gamma)$ is computed by traversing γ top-down and replacing every subtype τ of γ by $S_1(\tau) \cap \cdots \cap S_k(\tau)$ whenever $LV(\tau)$ intersects $\mathcal{V}_\psi(\Gamma, \sigma)$, i.e., for $Ex = Exp_{\langle \psi,k,\Gamma,\sigma \rangle}$,
   - $Ex(\tau_1 \cap \cdots \cap \tau_n) = \bigcap_{i=1}^{n} Ex(\tau_i)$
   - $Ex(\tau) = \bigcap_{i=1}^{k}(S_i(\tau))$, if $LV(\tau)$ intersects $\mathcal{V}_\psi(\Gamma, \sigma)$
   - $Ex(\gamma' \to \rho) = Ex(\gamma') \to Ex(\rho)$, if $LV(\rho)$ does not intersect $\mathcal{V}_\psi(\Gamma, \sigma)$
   - $Ex([\gamma']\rho) = [Ex(\gamma')]Ex(\rho)$, if $LV(\rho)$ does not intersect $\mathcal{V}_\psi(\Gamma, \sigma)$
   - $Ex(\mathtt{C}(\tau_1, \ldots, \tau_n)) = \mathtt{C}(Ex(\tau_1), \ldots, Ex(\tau_n))$, if $LV(\mathtt{C}(\tau_1, \ldots, \tau_n))$ does not intersect $\mathcal{V}_\psi(\Gamma, \sigma)$
   - $Ex(\varphi) = \varphi$, if φ is not in $\mathcal{V}_\psi(\Gamma, \sigma)$.

Example 3.12. When a polymorphic type is expected in the domain of an arrow or abstraction type, the intersection can be used without problems because our grammar allows intersection on the left-hand side of such types. However, if the polymorphism is expected in the right-hand side, i.e. in the resulting type, some special treatment is necessary. For example, to expand φ_1 in $[\varphi_1]\varphi_2$ one could obtain $[\varphi_3 \cap \varphi_4]\varphi_2$. On the other hand, to expand φ_2 in the same type, it is necessary to expand the entire type since we cannot add an intersection in the right-hand side. So the result would be $[\varphi_3]\varphi_5 \cap [\varphi_4]\varphi_6$.

The next lemma is a technical result used to prove the compatibility of type expansion with the ordering $\leq$.

Lemma 3.13. *For $\tau, \rho \in \mathcal{T}_s$, $\tau \leq \rho$ implies $LV(\tau) = LV(\rho)$.*

Proof. By induction on the derivation of $\tau \leq \rho$. We analyse the cases of transitivity and abstraction.

($\leq_{\mathbf{Trans}}$) In this case, there exists σ such that $\tau \leq \sigma \leq \rho$. By Lemma 3.4(1), $\sigma = \bigcap_{i=1}^{k} \sigma_i$ such that $\tau \leq \sigma_i$, for all $i = 1, \ldots, k$, and $\sigma_i \leq \rho$, for some i. Thus, by IH, it follows that $LV(\tau) = LV(\sigma_i)$, for all i. In this way, $LV(\sigma_i) = LV(\rho)$ for all i, by IH and because all $LV(\sigma_i)$ are equal. The result follows by transitivity.

($\leq_{\mathbf{abs}}$) $\tau = [\sigma]\tau'$, $\rho = [\gamma]\rho'$ and the following assertions are valid: $\gamma \leq \sigma$ and $\tau' \leq \rho'$. By IH, $LV(\tau') = LV(\rho')$. So, by definition of last type variable set, $LV(\tau) = LV(\rho)$.

□

Lemma 3.14 and its proof are also presented in Bakel and Fernández (1997) and it states the compatibility of expansion with the ordering $\leq$.

Lemma 3.14 (Compatibility of Expansion with $\leq$)**.** *Let $E = Exp_{\langle \psi, k, \Gamma, \sigma \rangle}$ be an expansion.*

1. *$\sigma \leq \gamma$ implies $E(\sigma) \leq E(\gamma)$.*
2. *$\Phi \leq \Gamma$ implies $E(\Phi) \leq E(\Gamma)$.*

Proof. The proof is by induction on the derivation of $\sigma \leq \gamma$. Only the case ($\leq_{\mathbf{abs}}$) will be presented here.

We have $\sigma = [\sigma']\rho$ and $\gamma = [\gamma']\tau$. As premises of the rule, it holds that $\gamma' \leq \sigma'$ and $\rho \leq \tau$. By Lemma 3.13, we have only two cases:

- If $LV([\sigma']\rho)$ and $LV([\gamma']\tau)$ intersect $\mathcal{V}_\psi(\Gamma, \sigma)$, then $E([\sigma']\rho) = \bigcap_{i=1}^{k} S_i([\sigma']\rho)$ and $E([\gamma']\tau) = \bigcap_{i=1}^{k} S_i([\gamma']\tau)$. Hence, by Lemma 3.9, $S_i([\sigma']\rho) \leq S_i([\gamma']\tau)$ for all $i = 1, \ldots, k$, and the result follows.
- If $LV([\sigma']\rho)$ does not intersect $\mathcal{V}_\psi(\Gamma, \sigma)$ so neither does $LV([\gamma']\tau)$ and vice versa. By IH, $E(\gamma') \leq E(\sigma')$ and $E(\rho) \leq E(\tau)$, what implies that $E([\sigma']\rho) = [E(\sigma')]E(\rho) \leq [E(\gamma')]E(\tau) = E([\gamma']\tau)$, as it was supposed to be proven.

□

4. Type Inference System and Basic Properties

This section presents type assignment rules for nominal terms, and explores fundamental properties of the type assignment system, such as the preservation of types for α-equivalent terms.

Definition 4.1. A **type judgement** is a tuple of a type environment, a term and a type, denoted by $\Gamma \vdash t : \gamma$. A type judgement $\Gamma \vdash t : \gamma$ is **derivable** if it can be deduced following the rules in Table 3; if so, then $\langle \Gamma, \gamma \rangle$ is called a **typing** of t. In Table 3, all types are strict except for the σ's that can be intersection types while $\mathcal{C}$ ranges over **chains of type operations**, that is a sequence $\langle O_1, \ldots, O_k \rangle$ of type operations of lifting, substitution and expansion that apply to types as follows:

$$\mathcal{C}(\sigma) = \langle O_1, \ldots, O_k \rangle(\sigma) = O_1(\ldots(O_k(\sigma))\ldots).$$

The empty chain is denoted by Id.

Remark 4.2. In Bakel (2011), a survey about intersection types is presented and different notions of chains are defined. Linear, relevant and essential chains are expressions used to designate chains where the operations must be applied in a specific order. For instance, in an essential chain, a lifting may occur at most once at the end of the chain. However, the current work follows a more general notion of chain, as presented in Bakel and Fernández (1997), where liftings, substitutions and expansions do not have a predetermined order in the sense that any kind of such operations can occur in any position of those sequences.

The rule $(\mathcal{T}_\cap)$ is actually an axiom if $k = 0$. Hence, any term is typable with type ω.

Table 3: Type System

$$(\mathcal{T}_{\mathbf{a}})\ \frac{\sigma \leq \tau}{\Gamma \bowtie a : \sigma \vdash a : \tau} \qquad (\mathcal{T}_{\mathbf{X}})\ \frac{\sigma \leq \tau}{\Gamma \bowtie X : \sigma \vdash \pi \cdot X : \tau}$$

$$(\mathcal{T}_{[\mathbf{a}]})\ \frac{\Gamma \bowtie a : \sigma \vdash t : \tau}{\Gamma \vdash [a]t : [\sigma]\tau} \qquad (\mathcal{T}_{\cap})\ \frac{\Gamma \vdash t : \tau_1 \quad \dots \quad \Gamma \vdash t : \tau_k}{\Gamma \vdash t : \tau_1 \cap \cdots \cap \tau_k}, k \neq 1$$

$$(\mathcal{T}_{\mathsf{f}})\ \frac{\sigma_1 \to \cdots \to \sigma_n \to \tau = \mathcal{C}(\Sigma_{\mathsf{f}}) \qquad \Gamma \vdash t_1 : \sigma_1 \quad \dots \quad \Gamma \vdash t_n : \sigma_n}{\Gamma \vdash \mathsf{f}(t_1, \dots, t_n) : \tau}$$

Example 4.3. Consider the term $\prod_{i=m}^{k} X$ with the same signature as in Example 3.2 and $\Gamma = \{m : \mathtt{nat}, k : \mathtt{nat}, X : \varphi\}$.

$$(\mathcal{T}_{\mathsf{f}})\ \frac{(\mathcal{T}_{[\mathbf{a}]})\ \dfrac{(\mathcal{T}_{\mathbf{X}})\ \dfrac{}{\Gamma \bowtie i : \mathtt{nat} \vdash X : \varphi}}{\Gamma \vdash [i]X : [\mathtt{nat}]\varphi} \qquad (\mathcal{T}_{\mathbf{a}})\ \frac{}{\Gamma \vdash m : \mathtt{nat}} \qquad (\mathcal{T}_{\mathbf{a}})\ \frac{}{\Gamma \vdash k : \mathtt{nat}}}{\Gamma \vdash \prod([i]X, m, k) : \varphi}$$

Notice that, if Γ had some type annotation for i, the same derivation would be still valid, since the judgement before rule $(\mathcal{T}_{[\mathbf{a}]})$ is updated with the proper annotation for this atom.

The next lemma is an inversion lemma on typing $\langle \Gamma, \sigma \rangle$ with respect to the structure of the corresponding term t.

Lemma 4.4 (Generation Lemma)**.**

1. *If $\Gamma \vdash a : \sigma$, then $\sigma = \omega$ or there exists $a : \gamma \in \Gamma$ such that $\gamma \leq \sigma$.*
2. *If $\Gamma \vdash \pi \cdot X : \sigma$, then $\sigma = \omega$ or there exists $X : \gamma \in \Gamma$ such that $\gamma \leq \sigma$.*
3. *If $\Gamma \vdash [a]t : \sigma$, then $\sigma = \bigcap_{i=1}^{n} [\rho_i]\tau_i$ and $\Gamma \bowtie a : \rho_i \vdash t : \tau_i$.*
4. *If $\Gamma \vdash \mathsf{f}(t_1, \dots, t_n) : \sigma$, then $\sigma = \bigcap_{j=1}^{k} \rho_j$ and there are chains of operations $\mathcal{C}_j$, $1 \leq j \leq k$, such that $\mathcal{C}_j(\Sigma_{\mathsf{f}}) = \gamma_{j_1} \to \cdots \to \gamma_{j_n} \to \rho_j$,*

$$\Gamma \vdash t_1 : \gamma_{j_1}\ ,\ \dots\ ,\ \Gamma \vdash t_n : \gamma_{j_n},\ \ \forall j = 1, \dots, k.$$

Proof. 1. The proof is by induction on the type derivation. The only rules that can be used in the last step of the derivation are $(\mathcal{T}_{\mathbf{a}})$ or $(\mathcal{T}_{\cap})$. If it is $(\mathcal{T}_{\mathbf{a}})$ so the statement is obtained because there is $a : \gamma$ in Γ with $\gamma \leq \sigma$. If the rule is $(\mathcal{T}_{\cap})$, then $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ and the deductions $\Gamma \vdash a : \sigma_i$ are valid for all $1 \leq i \leq n$. If $n = 0$ then $\sigma = \omega$. If $n \neq 0$, the IH can be used and there is a type annotation $a : \gamma \in \Gamma$ such that $\gamma \leq \sigma_i$, which implies that $\gamma \leq \sigma$ by rule $(\leq_{\cap\mathbf{I}})$. Notice that the same Γ is used in all the n subderivations.

2. The proof is similar to the previous item, but the rules used in the last step of the derivation can only be $(\mathcal{T}_{\mathbf{X}})$ or $(\mathcal{T}_{\cap})$.

3. The rules that can be used in the last step are $(\mathcal{T}_{[\mathbf{a}]})$ or $(\mathcal{T}_{\cap})$. If the rule is $(\mathcal{T}_{[\mathbf{a}]})$, then $\sigma = [\rho]\tau$. If the rule is $(\mathcal{T}_{\cap})$, then $\sigma = \delta_1 \cap \cdots \cap \delta_n$ and the deductions $\Gamma \vdash [a]t : \delta_i$ hold for every $1 \leq i \leq n$. Since each δ_i is a strict type, the rule $(\mathcal{T}_{\cap})$ cannot be applied again. So, each $\delta_i = [\rho_i]\tau_i$, as analysed first.

4. The only possible rules to apply are $(\mathcal{T}_{\mathsf{f}})$ and $(\mathcal{T}_{\cap})$. If $(\mathcal{T}_{\cap})$ is applied, then $\sigma = \bigcap_{j=1}^{k} \rho_j$, with k derivations. Since each ρ_j is a strict type, only the $(\mathcal{T}_{\mathsf{f}})$ rule could be applied before. So, the conditions for each ρ_j are supplied, as in the statement of the lemma.

□

Definition 4.5. A pair $\langle \Pi, \rho \rangle$ of an environment and a type is called a **principal pair** for a term t if it is a typing and, for every other typing $\langle \Gamma, \sigma \rangle$, there exists a chain of operations $\mathcal{C}$ such that $\mathcal{C}(\langle \Pi, \rho \rangle) = \langle \Gamma, \sigma \rangle$.

An important feature of this system is that a term may not have a principal pair, as the next example demonstrates.

Example 4.6. If the declaration of $\prod$ is $\Sigma_{\prod} = [\mathtt{nat}]\mathtt{real} \to \mathtt{nat} \to \mathtt{nat} \to \mathtt{real} \cap [\mathtt{nat}]\mathtt{rat} \to \mathtt{nat} \to \mathtt{nat} \to \mathtt{rat} \cap [\mathtt{nat}]\mathtt{nat} \to \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$, then the principal pair of $\prod_{i=Y}^{Z} X$ (sugar of $\prod([i]X, Y, Z)$) does not exist. Notice that $\langle \{X : \mathtt{real}, Y : \mathtt{nat}, Z : \mathtt{nat}\}, \mathtt{real} \rangle$ is a typing of this term and that substitutions and expansions do not change the type declaration, because it has no type variable. Then the least type with respect to $\leq$ that can be derived for $\prod_{i=Y}^{Z} X$ is $\mathtt{real} \cap \mathtt{rat} \cap \mathtt{nat}$. However, there is no type annotation of X that could derive such type and be greater than $\mathtt{real}$ with respect to $\leq$.

On the other hand, if $\Sigma_\Pi = [\mathtt{nat}]\varphi \to \mathtt{nat} \to \mathtt{nat} \to \varphi$, as in Example 3.2, the principal pair of the term is the typing $\langle\{X : \varphi, Y : \mathtt{nat}, Z : \mathtt{nat}\}, \varphi\rangle$.

Next, the operations of lifting, substitution and expansion are proved to be sound in the type system.

Lemma 4.7 (Soundness of Lifting)**.** *If $\Phi \vdash t : \tau$ and $L = L_{\langle\langle\Gamma,\tau\rangle,\langle\Gamma',\tau'\rangle\rangle}$ is a lifting, then $L(\Phi) \vdash t : \tau'$.*

Proof. The proof is by induction on the derivation of $\Phi \vdash t : \tau$. By definition of lifting, $\Gamma' \leq \Gamma$ and $\tau \leq \tau'$.

- Rule $(\mathcal{T}_{\mathbf{a}})$: If $a : \Phi_a \in \Gamma$, then a is annotated in Γ' and so it is in $L(\Phi)$. So, the relations $\Gamma'_a \leq \Phi_a \leq \tau \leq \tau'$ imply $L(\Phi) \vdash a : \tau'$. If $a : \Phi_a \notin \Gamma$, then $a : \Phi_a$ is kept unchanged in $L(\Phi)$. In this way, the relations $\Phi_a \leq \tau \leq \tau'$ likewise give us $L(\Phi) \vdash a : \tau'$.

- Rule $(\mathcal{T}_{\mathbf{X}})$: similar to the previous item.

- Rule $(\mathcal{T}_{[\mathbf{a}]})$: We have $\Phi \bowtie a : \sigma \vdash t' : \rho$. If $[\sigma]\rho \leq \tau'$, then $\tau' = \bigcap_{i=1}^{m} [\sigma'_i]\rho'_i$ by Lemma 3.4(2) and, for each i, $\sigma'_i \leq \sigma$ and $\rho \leq \rho'_i$. This implies that $L_i = L_{\langle\langle\Gamma\bowtie a:\sigma,\rho\rangle,\langle\Gamma'\bowtie a:\sigma'_i,\rho'_i\rangle\rangle}$ are well defined liftings and that $L_i(\Phi \bowtie a : \sigma) = L(\Phi) \bowtie a : \sigma'_i$. So, by IH, $L(\Phi) \bowtie a : \sigma'_i \vdash t' : \rho'_i$. Thus, using rules $(\mathcal{T}_{[\mathbf{a}]})$ and $(\mathcal{T}_\cap)$, we obtain $L(\Phi) \vdash [a]t' : \tau'$.

- Rule $(\mathcal{T}_\cap)$: $\tau = \tau_1 \cap \cdots \cap \tau_n \leq \tau'$, $\Phi \vdash t : \tau_j$ for each $j = 1, \ldots, n$ and, by Lemma 3.4(1), $\tau' = \bigcap_{i=1}^{m} \tau'_i$ such that, for all $i = 1, \ldots, m$, there exists $j_i = 1, \ldots, n$ satisfying $\tau_{j_i} \leq \tau'_i$. Define the liftings $L_i = L_{\langle\langle\Gamma,\tau_{j_i}\rangle,\langle\Gamma',\tau'_i\rangle\rangle}$, for $i = 1, \ldots, m$; notice that $L_i(\Phi) = L(\Phi)$. By IH, $L(\Phi) \vdash t : \tau'_i$ and, applying $(\mathcal{T}_\cap)$, one obtains $L(\Phi) \vdash t : \tau'$.

- Rule $(\mathcal{T}_{\mathsf{f}})$: There exists a chain $\mathcal{C}$ of operations such that $\mathcal{C}(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \tau$ and $\Phi \vdash t_i : \sigma_i$, for all $i = 1, \ldots, n$. Define $L_i = L_{\langle\langle\Gamma,\sigma_i\rangle,\langle\Gamma',\sigma_i\rangle\rangle}$ and observe that $L_i(\Phi) = L(\Phi)$. By IH, we obtain $L(\Phi) \vdash t_i : \sigma_i$, for all $i = 1, \ldots, n$. Since $\sigma_1 \to \cdots \to \sigma_n \to \tau \leq \sigma_1 \to \cdots \to \sigma_n \to \tau'$ (because $\tau \leq \tau'$), the lifting $L' = L_{\langle\langle\varnothing,\sigma_1\to\cdots\to\sigma_n\to\tau\rangle,\langle\varnothing,\sigma_1\to\cdots\to\sigma_n\to\tau'\rangle\rangle}$ is well defined and $L' \circ \mathcal{C}(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \tau'$, which gives us $L(\Phi) \vdash \mathsf{f}(t_1, \ldots, t_n) : \tau'$.

□

Lemma 4.8 (Soundness of type substitution)**.** *Let S be a type substitution and $\Gamma \vdash t : \sigma$ a derivable judgement. Then, $S(\Gamma) \vdash t : S(\sigma)$ has a type derivation that is exactly as the original with the substitution S applied in each node.*

Proof. Since the case where $S(\sigma) = \omega$ is trivial, we consider $S(\sigma) \neq \omega$. The proof proceeds by induction on the derivation of $\Gamma \vdash t : \sigma$.

- Rule $(\mathcal{T}_{\mathbf{a}})$: $t = a$ and $\Gamma_a \leq \sigma$. Since $S(\Gamma_a) \leq S(\sigma)$ by Lemma 3.9, we obtain $S(\Gamma) \vdash a : S(\sigma)$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: Similar to the previous item.
- Rule$(\mathcal{T}_{[\mathbf{a}]})$: $t = [a]t'$, $\sigma = [\gamma]\tau$ and $\Gamma \bowtie a : \gamma \vdash t' : \tau$. By IH, $S(\Gamma) \bowtie a : S(\gamma) \vdash t' : S(\tau)$. Since $S([\gamma]\tau) \neq \omega$, one has $S[\tau] \neq \omega$ and $S([\gamma]\tau) = [S(\gamma)]S(\tau)$. In this way, $S(\Gamma) \vdash [a]t' : S([\gamma]\tau)$.
- Rules $(\mathcal{T}_{\cap})$ and $(\mathcal{T}_{\mathsf{f}})$: these cases are developed in Bakel and Fernández (1997).

□

Lemma 4.9 (Soundness of expansion)**.** *Let E be an expansion and $\Gamma \vdash t : \sigma$ a derivable judgement. Then, $E(\Gamma) \vdash t : E(\sigma)$ is also derivable.*

Proof. Suppose w.l.o.g. that $\sigma \in \mathcal{T}_s$. The proof of Theorem 4.4.3 in Bakel and Fernández (1997) covers most of the cases, including the case where $E(\sigma) \notin \mathcal{T}_s$. We only need to add the case of rule $(\mathcal{T}_{[\mathbf{a}]})$ where $E(\sigma) \in \mathcal{T}_s$.

In such case, one has $\sigma = [\sigma']\tau$, $t = [a]t'$ and $\Gamma \bowtie a : \sigma' \vdash t' : \tau$. By IH, we obtain $E(\Gamma) \bowtie a : E(\sigma') \vdash t' : E(\tau)$. Applying rule (abs), we obtain $E(\Gamma) \vdash [a]t' : [E(\sigma')]E(\tau)$. If the last type variable set of τ intersected $\mathcal{V}_\psi(\Gamma, \sigma)$, $E([\sigma']\tau)$ would be an intersection, what contradicts $E(\sigma) \in \mathcal{T}_s$. So, $E([\sigma']\tau) = [E(\sigma')]E(\tau)$, which concludes the proof. □

The proposition that follows states that atoms that are fresh in a term are not relevant in a typing (they can be removed or updated in the environment).

Proposition 4.10 (Type weakening and strengthening)**.** *The following properties hold:*

1. *If $\Gamma \vdash t : \sigma$ and there exists some Δ such that $\Delta \vdash a\#t$, then $\Gamma \bowtie a : \tau \vdash t : \sigma$. (Type weakening).*

2. *If* $\Gamma \bowtie a : \tau \vdash t : \sigma$ *and there exists some* Δ *such that* $\Delta \vdash a\#t$*, then* $\Gamma \vdash t : \sigma$*. (Type strengthening)*

Proof. The proof is by induction on derivations.

1. - Rule $(\mathcal{T}_{\mathbf{a}})$: $\Gamma \bowtie b : \gamma \vdash b : \sigma$ with $\gamma \leq \sigma$, $a \notin atms(b)$ and $\Delta \vdash a\#b$. So, $\Gamma \bowtie b : \gamma \bowtie a : \tau \vdash b : \sigma$ using rule $(\mathcal{T}_{\mathbf{a}})$.
   - Rule $(\mathcal{T}_{\mathbf{X}})$: similarly to the previous case, $\Gamma \bowtie X : \gamma \bowtie a : \tau \vdash \pi \cdot X : \sigma$ with $\gamma \leq \sigma$.
   - Rule $(\mathcal{T}_{[\mathbf{a}]})$: $t = [b]t'$, $\sigma = [\sigma']\sigma''$ and $\Gamma \bowtie b : \sigma' \vdash t' : \sigma''$. One has to consider 2 cases. If $a = b$, then $\Gamma \bowtie a : \tau \bowtie a : \sigma' = \Gamma \bowtie b : \sigma'$ and we have the same valid judgement. If $a \neq b$, then $\Delta \vdash a\#t'$ (or $a \notin atms(t')$) and, by IH, $\Gamma \bowtie b : \sigma' \bowtie a : \tau \vdash t' : \sigma''$.
   - Rule $(\mathcal{T}_{\cap})$: it can be obtained applying IH in the cases that $\sigma \neq \omega$. If $\sigma = \omega$, the derivation is trivial.
   - Rule $(\mathcal{T}_{\mathsf{f}})$: it holds directly applying the induction hypothesis.
2. The proof for strengthening is similar to the weakening proof.

□

The following lemma will be used in the context of "typed rewrite steps" in Section 5, where renamed versions of rules are allowed and meta-level equivariance is needed. Moreover, the object-level equivariance explains why we do not check the types of the atoms in permutations in the rule $(\mathcal{T}_{\mathbf{X}})$: permutations do not change types, whenever the same renaming is used in the type environment.

Lemma 4.11 (Meta-Level and Object-Level Equivariance of Type Derivations)**.** *The following statements are equivalent:*

1. $\Gamma \vdash t : \tau$ *is derivable;*
2. ${}^{\pi}\Gamma \vdash {}^{\pi}t : \tau$ *is derivable, where the permutation acts the same way throughout the derivation, applied to each node;*
3. ${}^{\pi}\Gamma \vdash \pi \bullet t : \tau$ *is derivable, where the permutation acts the same way throughout the derivation, applied to each node.*

Proof. • **Equivalence between** 1 **and** 2**:** By induction on type derivations.

- Rule $(\mathcal{T}_{\mathbf{a}})$: If $\sigma \leq \tau$, then $\Gamma \bowtie a : \sigma \vdash a : \tau$ if and only if ${}^{\pi}\Gamma \bowtie \pi(a) : \sigma \vdash {}^{\pi}a : \tau$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: the annotation of variables is not changed by permutations.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: By IH, $\Gamma \bowtie a : \sigma \vdash t' : \rho$ if and only if ${}^{\pi}\Gamma \bowtie \pi(a) : \sigma \vdash {}^{\pi}t' : \rho$ and the result follows by rule $(\mathcal{T}_{[\mathbf{a}]})$.
- Rule$(\mathcal{T}_{\cap})$: By IH, $\Gamma \vdash t : \tau_i$ for all $i = 1, \ldots, m$ if and only if ${}^{\pi}\Gamma \vdash {}^{\pi}t : \tau_i$ and this case is finished by applying rule $(\mathcal{T}_{\cap})$.
- Rule $(\mathcal{T}_{\mathsf{f}})$: There is a chain $\mathcal{C}$ such that $\mathcal{C}(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \tau$ and, for all $i = 1, \ldots, n$, $\Gamma \vdash t_i : \sigma_i$. By IH, ${}^{\pi}\Gamma \vdash {}^{\pi}t_i : \sigma_i$. So, ${}^{\pi}\Gamma \vdash {}^{\pi}\mathsf{f}(t_1, \ldots, t_n) : \tau$. The reverse is analogous.

- **Equivalence between** 1 **and** 3**:** Similar to the previous equivalence.

□

Example 4.12. Consider $\Gamma = \{X : \mathtt{nat}, a : \mathtt{nat}\}$ and the derivable type judgement $\Gamma \vdash +(a, X) : \mathtt{nat}$, where $\Sigma_{+} = \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$. It is also possible to derive ${}^{(a\,b)}\Gamma \vdash +(b, (a\,b) \cdot X) : \mathtt{nat}$ by object level equivariance, and ${}^{(a\,b)}\Gamma \vdash +(b, X) : \mathtt{nat}$ by meta-level equivariance. We show the derivation for the first one.

$$(\mathcal{T}_{\mathsf{f}})\ \frac{(\mathcal{T}_{\mathbf{a}})\ \dfrac{}{\Gamma \vdash a : \mathtt{nat}} \qquad (\mathcal{T}_{\mathbf{X}})\ \dfrac{}{\Gamma \vdash X : \mathtt{nat}}}{\Gamma \vdash +(a, X) : \mathtt{nat}}$$

$$\updownarrow$$

$$\frac{\dfrac{}{{}^{(a\,b)}\Gamma \vdash b : \mathtt{nat}}\ (\mathcal{T}_{\mathbf{a}}) \qquad \dfrac{}{{}^{(a\,b)}\Gamma \vdash (a\,b) \cdot X : \mathtt{nat}}\ (\mathcal{T}_{\mathbf{X}})}{{}^{(a\,b)}\Gamma \vdash +(b, (a\,b) \cdot X) : \mathtt{nat}}\ (\mathcal{T}_{\mathsf{f}})$$

Lemma 4.13 asserts that two α-equivalent terms have the same typings, which is expected since such terms represent the same class of objects.

Lemma 4.13 (α-equivalence preserves types)**.** *If $\Gamma \vdash t : \sigma$ is derivable and $\Delta \vdash t \approx_{\alpha} s$ for some Δ, then $\Gamma \vdash s : \sigma$ is also derivable.*

Proof. By induction on the derivation of α-equivalence.

- Rule $(\approx_{\alpha}\mathbf{a})$: $t = s = a$: it is straightforward.

- Rule $(\approx_\alpha \mathbf{X})$: $t = \pi\cdot X$ and $s = \pi'\cdot X$: since $\Gamma \vdash \pi\cdot X : \sigma$, by the Generation Lemma (4.4), there exists γ such that $X : \sigma \in \Gamma$ and $\gamma \leq \sigma$.Thus, $\Gamma \vdash \pi' \cdot X : \sigma$ by applying rule $(\mathcal{T}_{\mathbf{X}})$.

- Rule $(\approx_\alpha [\mathbf{a}])$ or $(\approx_\alpha [\mathbf{b}])$: $t = [a]t'$ and $s = [b]s'$: we know that $\sigma = \cap_{i=1}^{n}[\sigma_i]\gamma_i$ and that $\Gamma \bowtie a : \sigma_i \vdash t' : \gamma_i$ for all $i = 1, \ldots, n$, by the Generation Lemma (4.4). If $a = b$, then $\Delta \vdash t' \approx_\alpha s'$ and, by IH, $\Gamma \bowtie b : \sigma_i \vdash s' : \gamma_i$ and it follows that $\Gamma \vdash [b]s' : \tau$. If $a \neq b$, then $\Delta \vdash t' \approx_\alpha (a\,b)\bullet s', a\#s', (a\,b)\bullet t' \approx_\alpha s', b\#t'$. Let's consider the more complex case when $a, b \in \Gamma$. Thus,

$\Gamma \setminus \{b : \Gamma_b\} \bowtie a : \sigma_i \vdash t' : \gamma_i$,	by Lemma 4.10(2)–strengthening,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \vdash t' : \gamma_i$,	i.e., the same judgement,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \bowtie b : \sigma_i \vdash t' : \gamma_i$,	by Lemma 4.10(1)–weakening,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \bowtie b : \sigma_i \vdash (a\,b)\bullet t' : \gamma_i$,	by Lemma 4.11
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \bowtie b : \sigma_i \vdash s' : \gamma_i$,	by induction hypothesis,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \vdash [b]s' : [\sigma_i]\gamma_i$,	applying rule $(\mathcal{T}_{[\mathbf{a}]})$,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \vdash [b]s' : [\sigma_i]\gamma_i$,	by Lemma 4.10(2),
$\Gamma\setminus \vdash [b]s' : [\sigma_i]\gamma_i$,	by Lemma 4.10(1),
$\Gamma\setminus \vdash [b]s' : \sigma$,	applying rule $(\mathcal{T}_{\cap})$.

- Rule $(\approx_\alpha \mathsf{f})$: $t = \mathsf{f}(t_1, \ldots, t_n)$: this case follows by item 4 of Generation Lemma (4.4) and IH, applying rules $(\mathcal{T}_{\mathsf{f}})$ and $(\mathcal{T}_{\cap})$ at the end.

□

Example 4.14. Consider a signature where $\Sigma_\forall = ([\varphi]\mathtt{bool}) \to \mathtt{bool}$, $\Sigma_{\mathsf{even}} = \mathtt{nat} \to \mathtt{bool}$ and $\Sigma_+ = \varphi \to \varphi \to \varphi$ (+ will be written infixed). By the following derivation, the term $\forall[a]\mathsf{even}(a + X)$ has typing $\langle\{X : \mathtt{nat}\}, \mathtt{bool}\rangle$.

$$
(\mathcal{T}_{\mathsf{f}})\ \dfrac{(\mathcal{T}_{[\mathbf{a}]})\ \dfrac{(\mathcal{T}_{\mathsf{f}})\ \dfrac{X : \mathtt{nat}, a : \mathtt{nat} \vdash a : \mathtt{nat}\ (\mathcal{T}_{\mathbf{a}}) \qquad X : \mathtt{nat}, a : \mathtt{nat} \vdash X : \mathtt{nat}\ (\mathcal{T}_{\mathbf{X}})}{X : \mathtt{nat}, a : \mathtt{nat} \vdash a + X : \mathtt{nat}}}{X : \mathtt{nat}, a : \mathtt{nat} \vdash \mathsf{even}(a + X) : \mathtt{bool}}}{X : \mathtt{nat} \vdash [a]\mathsf{even}(a + X) : [\mathtt{nat}]\mathtt{bool}}}{X : \mathtt{nat} \vdash \forall[a]\mathsf{even}(a + X) : \mathtt{bool}}
$$

The same typing is also valid for $\forall[b]\mathsf{even}(b + (a\,b)\cdot X)$, by Lemma 4.13, because $b\#X \vdash \forall[a]\mathsf{even}(a + X) \approx_\alpha \forall[b]\mathsf{even}(b + (a\,b)\cdot X)$.

The Subterm Lemma presented next shows that a derivation of a judgement whose type is different from ω contains typings for the subterms of the term in question.

Lemma 4.15 (Subterm Lemma)**.** *If $\Gamma \vdash t : \sigma$ is derivable with a derivation $\mathcal{D}$ without any subterm typed with ω and s is a proper subterm of t, then there is a subtree of $\mathcal{D}$ that derives $\Gamma' \vdash s : \gamma$ for some Γ' and γ such that Γ' is an extension of Γ.*

Proof. By induction on the type derivation of $\Gamma \vdash t : \sigma$.

- Rules $(\mathcal{T}_{\mathbf{a}})$ and $(\mathcal{T}_{\mathbf{X}})$ are trivial because there is no proper subterm of an atom or a suspended variable.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: We have $\Gamma \bowtie a : \rho \vdash t' : \delta$, $t = [a]t'$ and $\sigma = [\rho]\delta$. If $s = t'$, it is done. If not, then s is a proper subterm of t' and, by IH, there are Γ' and γ such that $\Gamma' \vdash s : \gamma$ is a subtree in the derivation of $\Gamma \bowtie a : \rho \vdash t' : \delta$.
- Rule $(\mathcal{T}_{\cap})$: $\sigma = \tau_1 \cap \cdots \cap \tau_k$ and, for each $i = 1, \ldots, k$, $\Gamma \vdash t : \tau_i$. By IH, there are Γ_i's and γ_i's such that $\Gamma_i \vdash s : \gamma_i$ is a subtree of the derivation of $\Gamma \vdash t : \tau_i$. In this branch, it is important to observe that $k \neq 0$; otherwise, the IH would not be possible.
- Rule $(\mathcal{T}_{\mathsf{f}})$: $t = \mathsf{f}(t_1, \ldots, t_n)$, there is a chain $\mathcal{C}$ such that $\mathcal{C}(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \sigma$ and $\Gamma \vdash t_i : \sigma_i$ for all $i = 1, \ldots, n$. If $s = t_i$ for some t_i, then the result follows. Otherwise, s is a proper subterm of some t_i and the result is obtained by IH.

□

5. Typed Matching and Typed Rewrite Relation

This section introduces the notion of typed matching, which will be used to define a rewrite relation that takes types into account. First, we introduce the notation used in the definition of typed matching, and motivate the inclusion of freshness contexts in matching problems.

Matching will instantiate type variables as well as unknowns, thus, the pattern of a matching problem should include a term in context, as in Fernández and Gabbay (2007), and a typing of such term. The term and type substitutions that solve the matching problem should satisfy the conditions expressed in each leaf with unknowns of the type derivation. The

notion of *variable environment* introduced next extracts the relevant information from a type derivation, eliminating the type annotations of atoms that are fresh for some unknown.

Definition 5.1. If $\Gamma \vdash \pi \cdot X : \sigma$ is a leaf in the type derivation $\mathcal{D}$ of $\Gamma' \vdash t : \gamma$, then ${}^{\pi^{-1}}\Gamma \setminus \{a : \rho \mid \Delta \vdash a\#X \text{ or } \rho = \omega\}$ is a **variable environment** of X in $\mathcal{D}$ under Δ.

Notice that atoms that cannot occur free in the instance of a variable are not included in a variable environment and neither are the atoms annotated with ω, because they do not add information.

Example 5.2. Let $[\mathtt{nat}]\varphi \to [\mathtt{string}]\varphi \to \mathtt{real}$ be the type declaration of a binary symbol $\mathtt{g}$. Consider the judgement $X : \mathtt{nat} \vdash \mathtt{g}([a]X, [a](a\,b) \cdot X) : \mathtt{real}$ with the following derivation:

$$\mathcal{D}_1 = \dfrac{X : \mathtt{nat}, a : \mathtt{string} \vdash (a\,b) \cdot X : \mathtt{nat}(\mathcal{T}_{\mathbf{X}})}{X : \mathtt{nat} \vdash [a](a\,b) \cdot X : [\mathtt{string}]\mathtt{nat}}\,(\mathcal{T}_{[\mathbf{a}]})$$

$$\mathcal{D} = \dfrac{\dfrac{X : \mathtt{nat}, a : \mathtt{nat} \vdash X : \mathtt{nat}(\mathcal{T}_{\mathbf{X}})}{X : \mathtt{nat} \vdash [a]X : [\mathtt{nat}]\mathtt{nat}}\,(\mathcal{T}_{[\mathbf{a}]}) \qquad \mathcal{D}_1}{X : \mathtt{nat} \vdash \mathtt{g}([a]X, [a](a\,b) \cdot X) : \mathtt{real}}\,(\mathcal{T}_{\mathbf{f}})$$

If we observe the leaf on the left-hand side of $\mathcal{D}$, then the substitution $[X \mapsto a]$ respects the environment $\{X : \mathtt{nat}, a : \mathtt{nat}\}$ maintaining the type in the instance of X. On the right-hand side, in the subderivation $\mathcal{D}_1$, the atom a is annotated with type $\mathtt{string}$ in the environment; applying the inverse of $(a\,b)$, which is itself, to the environment, we obtain $\{X : \mathtt{nat}, b : \mathtt{string}\}$, that is a variable environment of X in $\mathcal{D}$ under $\varnothing$. Notice that enlarging this environment with $a : \mathtt{nat}$ allows us to proceed with the instance of the term keeping the designated type.

In the case that the context is not empty, for instance $\{a\#X\}$, then the variable environments of X in $\mathcal{D}$ under $\{a\#X\}$ would be $\{X : \mathtt{nat}\}$ and $\{X : \mathtt{nat}, b : \mathtt{string}\}$, respectively.

Definition 5.3 (Typed Matching Problem)**.** Let $\Phi \vdash l : \tau$ be a type judgement with derivation $\mathcal{D}$, and let ∇, Δ be freshness environments, Γ a type environment, and s a term. A typed matching problem is a pair of the form $(\Phi \Vdash \nabla \vdash l : \tau)^{?}\approx_{\alpha} (\Gamma \Vdash \Delta \vdash s)$, with $unkn(\Phi, \nabla, l)$ and $Vars(\Phi, \tau)$ disjoint from $unkn(\Gamma, \Delta, s)$ and $Vars(\Gamma)$.

The typed matching problem $(\Phi \Vdash \nabla \vdash l : \tau)^? \approx_\alpha (\Gamma \Vdash \Delta \vdash s)$ has solution $(\mathcal{C}, \theta, \pi)$, where $\mathcal{C}$ is a chain of operations, θ a substitution and π a permutation, whenever:

1. $\Delta \vdash {}^{\pi}\nabla\theta,\ {}^{\pi}l\theta \approx_\alpha s$;
2. $\mathcal{C}({}^{\pi}\Phi)|_{\mathbb{A}} \subseteq \Gamma$, i.e., the atoms of Φ renamed with π have the annotations changed only by $\mathcal{C}$ in Γ; and
3. ${}^{\pi'^{-1}}\Gamma \bowtie \mathcal{C}(\Phi') \vdash X\theta : \mathcal{C}(\Phi)_X$, for every variable environment Φ' of X in ${}^{\pi}\Delta \vdash {}^{\pi}\mathcal{D}$, with π' the corresponding permutation in each leaf. Notice that $\Phi_X = \Phi'_X$ because renamings do not change annotations of unknowns.

Example 5.4. Take the symbol $\mathtt{g}$ as in Example 5.2. The following typed matching problem

$$(X : \mathtt{nat} \Vdash b\#X \vdash \mathtt{g}([a]X, [a](a\,b)\cdot X) : \mathtt{real})^? \approx_\alpha (b : \mathtt{nat} \Vdash \varnothing \vdash \mathtt{g}([b]b, [c]b))$$

has a solution $(Id, [X \mapsto a], id)$, i.e., $\varnothing \vdash b\#a,\ \mathtt{g}([a]a, [a]b) \approx_\alpha \mathtt{g}([b]b, [c]b)$ and, taking the variables environments in $\mathcal{D}$ under $\{b\#X\}$ (Example 5.2), $b : \mathtt{nat}, X : \mathtt{nat}, a : \mathtt{nat} \vdash a : \mathtt{nat}$ is derivable as well as $a : \mathtt{nat}, X : \mathtt{nat}, b : \mathtt{string} \vdash a : \mathtt{nat}$.

Example 5.5. Consider $\prod$ with the type declaration of Example 3.2 and take the binary symbol $/$ (with $\Sigma_/ = \mathtt{nat} \to \mathtt{nat} \to \mathtt{rat}$ and infix notation), the unary symbol $\mathtt{s}$ (with $\Sigma_\mathtt{s} = \mathtt{nat} \to \mathtt{nat}$) and the nullary symbol $\mathtt{0}$ (with $\Sigma_\mathtt{0} = \mathtt{nat}$). The following typed matching problem

$$\left(X : \mathtt{real} \cap \mathtt{rat} \cap \mathtt{nat}, Y : \mathtt{nat}, Z : \mathtt{nat} \Vdash i\#X \vdash \prod_{i=Y}^{Z} X : \mathtt{real} \cap \mathtt{rat} \cap \mathtt{nat}\right)^? \approx_\alpha$$

$$\left(m : \mathtt{nat} \Vdash \varnothing \vdash \prod_{i=1}^{m} \left(\frac{\mathsf{s(0)}}{\mathsf{s(s(0))}}\right)\right)$$

has solution:

$$\left(L_{\langle\langle\varnothing,\mathtt{real}\cap\mathtt{rat}\cap\mathtt{nat}\rangle,\langle\varnothing,\mathtt{rat}\rangle\rangle}, \left[X \mapsto \frac{\mathsf{s(0)}}{\mathsf{s(s(0))}}\right] \circ [Y \mapsto (s(0)] \circ [Z \mapsto m], id\right)$$

Observe that the instance of the term satisfies the freshness constraint of the pattern in the empty context.

The next lemma is inspired by the presentation given in Fairweather (2014).

Lemma 5.6 (Typed Nominal Pattern Matching)**.** *If* $\Phi \vdash l : \sigma$ *and* $(\Phi \Vdash \nabla \vdash l : \sigma)^{?}\approx_{\alpha} (\Gamma \Vdash \Delta \vdash s) = (\mathcal{C}, \theta, \pi)$*, then* $\Gamma \vdash {}^{\pi}l\theta : \mathcal{C}(\sigma)$*.*

Proof. Firstly, by meta-level equivariance (Lemma 4.11), soundness of type operations in $\mathcal{C}$ (Lemmas 4.7, 4.8 and 4.9) and weakening (Lemma 4.10(1)), $\Gamma \vdash {}^{\pi}l : \mathcal{C}(\sigma)$ is derivable. To include the term-substitution θ, we proceed by induction on the derivation of this last judgement. Assume that $\mathcal{C}(\sigma) \neq \omega$, otherwise the result follows directly using rule $(\mathcal{T}_{\cap})$.

- Rule $(\mathcal{T}_{\mathbf{a}})$: this part is trivial because θ has no effect on ${}^{\pi}l$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: ${}^{\pi}l = \pi' \cdot X$. By Condition 3 of Definition 5.3,

  $$ {}^{\pi'^{-1}}\Gamma \bowtie \mathcal{C}({}^{\pi'^{-1}\circ\pi}\Phi \setminus \{a : \rho \mid {}^{\pi}\nabla \vdash a\#X\}) \vdash X\theta : \mathcal{C}({}^{\pi}\Phi)_X, $$

  which implies that

  $$ \Gamma \bowtie \mathcal{C}({}^{\pi}\Phi \setminus \{a : \rho \mid {}^{\pi}\nabla \vdash a\#X\}) \vdash \pi' \cdot X\theta : \mathcal{C}({}^{\pi}\Phi)_X $$

  using object-level equivariance (Lemma 4.11). Since $\mathcal{C}({}^{\pi}\Phi)|_{\mathbb{A}} \subseteq \Gamma$ and $\mathcal{C}({}^{\pi}\Phi)_X \leq \mathcal{C}(\sigma)$ (by Lemmas 3.9 and 3.14 and the definition of lifting), the derivation of $\Gamma \vdash \pi' \cdot X\theta : \mathcal{C}(\sigma)$ follows easily by Lemma 4.7.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: $l = [a]l'$ and $\mathcal{C}(\sigma) = [\sigma']\tau$. We have $\Gamma \bowtie \pi(a) : \sigma' \vdash {}^{\pi}l' : \tau$ and, since $(\mathcal{C}, \theta, \pi)$ is also a solution for the matching problem $(\Phi \bowtie a : \sigma' \Vdash \nabla \vdash l' : \tau)^{?}\approx_{\alpha} (\Gamma \bowtie a : \sigma' \Vdash \Delta \vdash (a\,b) \bullet s')$ for $s = [b]s'$, so one has $\Gamma \bowtie \pi(a) : \sigma' \vdash {}^{\pi}l'\theta : \tau$ by IH. Applying rule $(\mathcal{T}_{[\mathbf{a}]})$, the result holds.
- Rule $(\mathcal{T}_{\cap})$: $\mathcal{C}(\sigma) = \tau_1 \cap \cdots \cap \tau_k$. It holds that $\Gamma \vdash {}^{\pi}t : \tau_i$. Since $\mathcal{C}(\sigma) \neq \omega$ and so $k \neq 0$, by IH, $\Gamma \vdash {}^{\pi}t\theta : \tau_i$ holds for each $i = 1, \ldots, k$. Applying rule $(\mathcal{T}_{\cap})$, $\Gamma \vdash {}^{\pi}t\theta : \mathcal{C}(\sigma)$ is obtained.
- Rule $(\mathcal{T}_{\mathsf{f}})$: $l = \mathsf{f}(l_1, \ldots, l_n)$, there is $\mathcal{C}'$ such that $\mathcal{C}'(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \mathcal{C}(\sigma)$ and $\Gamma \vdash {}^{\pi}l_i : \sigma_i$, for $i = 1, \ldots, n$. Since the environment is not changed in this step, by IH, we have $\Gamma \vdash {}^{\pi}l_i\theta : \sigma_i$. The result is obtained applying rule $(\mathcal{T}_{\mathsf{f}})$.

□

The previous lemma is important because it ensures that if a term matches a pattern, then it can be typed in the same way as the pattern by applying to the pattern type the chain of operations specified in the matching solution and with some additional type annotations (of fresh atoms and/or new unknowns). Notice that, by Definition 5.3, such type checking is not necessary when verifying if a triplet is a solution of the typed matching problem.

The next lemma states that if an instance of a term with new unknowns can be typed, the term can also be typed with the same type if we add in the environment the annotations for its unknowns.

Lemma 5.7 (Inversion Substitution Lemma)**.** *Let $\Gamma \vdash t\theta : \gamma$ be a derivable judgement such that $unkn(t) \cap unkn(t\theta) = \varnothing$. Then, there is an environment Γ^* which is Γ updated with annotations of variables in $unkn(t)$ and such that $\Gamma^* \vdash t : \gamma$ is derivable.*

Proof. The proof is by induction on the derivation of $\Gamma \vdash t\theta : \gamma$, but first we consider the case when $t = \pi \cdot X$. In this case, $\Gamma \bowtie X : \gamma \vdash \pi \cdot X : \gamma$ is derivable. Now, assume that t is not a suspended variable.

- Rule $(\mathcal{T}_{\mathbf{a}})$: $t\theta = a$ and, since t is not a variable, $t = a$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: the only way this happens is when t is a suspended variable because the variables are different from the ones of $t\theta$.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: $t = [a]t'$, $t\theta = [a]t'\theta$, $\gamma = [\sigma]\tau$ and $\Gamma \bowtie a : \sigma \vdash t'\theta : \tau$. By IH, there is Γ^* such that $\Gamma^* \bowtie a : \sigma \vdash t' : \tau$ and, applying rule $(\mathcal{T}_{[\mathbf{a}]})$, one obtains $\Gamma^* \vdash [a]t' : [\sigma]\tau$.
- Rule $(\mathcal{T}_{\cap})$: for $i = 1, \ldots, k$, $\Gamma \vdash t\theta : \tau_i$ and $\gamma = \tau_1 \cap \cdots \cap \tau_k$. By IH, $\Gamma^* \vdash t : \tau_i$ and, applying rule $(\mathcal{T}_{\cap})$, $\Gamma^* \vdash t : \gamma$.
- Rule $(\mathcal{T}_{\mathsf{f}})$: $t = \mathsf{f}(t_1, \ldots, t_n)$, $\Gamma \vdash t_i\theta : \sigma_i$ for $i = 1, \ldots, n$ and there exists $\mathcal{C}$ such that $\mathcal{C}(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \gamma$. By IH, there is Γ_i with annotations of variables in $unkn(t_i)$ and such that $\Gamma \bowtie \Gamma_i \vdash t_i : \sigma_i$. Build Γ' as the intersection of annotations of unknowns in Γ_i, for all $i = 1, \ldots, n$, and take $\Gamma^* = \Gamma \bowtie \Gamma'$. This environment satisfies $\Gamma^* \vdash t_i : \sigma_i$, by soundness of lifting (Lemma 4.7), since $\Gamma^* \leq \Gamma \bowtie \Gamma_i$. Using rule (f), one has $\Gamma^* \vdash \mathsf{f}(t_1, \ldots, t_n) : \gamma$.

□

Definition 5.8. A **typed rewrite rule** $\Phi \Vdash \nabla \vdash l \to r : \rho$ is a tuple of an environment, a freshness context, two terms and a type such that $\langle \Phi, \rho \rangle$ is a principal pair for l and it is a typing for r, in such a way that the variable environments of each unknown are equal in the derivations of $\Phi \vdash l : \rho$ and of $\Phi \vdash r : \rho$. Additionally, $unkn(\Phi, \nabla, r) \subseteq unkn(l)$.

The condition of having equal variable environments throughout derivations is called the diamond property in Fairweather (2014) and the compatibility property in Fairweather and Fernández (2018). There, only derivations with this property are valid. Here, it is required only for rewrite rules in order to obtain the Subject Reduction Lemma, as proved in Subsection 5.1.

Example 5.9. The rules in the system of Table 4 are typed rewrite rules. Take, for instance, the fifth rule. The type derivation for the left-hand side is:

$$\dfrac{\dfrac{\dfrac{\dfrac{X : \varphi, a : \omega, b : \omega \vdash X : \varphi \ (\mathcal{T}_{\mathbf{X}})}{X : \varphi, a : \omega \vdash [b]X : [\omega]\varphi} \ (\mathcal{T}_{[\mathbf{a}]})}{X : \varphi, a : \omega \vdash \mathsf{Lam}[b]X : \omega \to \varphi} \ (\mathcal{T}_{\mathsf{f}})}{X : \varphi \vdash [a]\mathsf{Lam}[b]X : [\omega](\omega \to \varphi)} \ (\mathcal{T}_{[\mathbf{a}]}) \qquad X : \varphi \vdash Z : \omega \ (\mathcal{T}_{\cap})}{X : \varphi \vdash \mathsf{Sub}([a]\mathsf{Lam}[b]X, Z) : \omega \to \varphi} \ (\mathcal{T}_{\mathsf{f}})$$

Notice that this typing is a principal pair of $\mathsf{Sub}([a]\mathsf{Lam}[b]X, Z)$ and so is $\langle \{X : \varphi\}, \varphi' \to \varphi \rangle$ because each of them can be obtained by the other and they generate any other typing for this term. Applying the lifting $L_{\langle \langle \{X:\varphi\}, \omega \to \varphi \rangle, \langle \{X:\varphi\}, \varphi' \to \varphi \rangle \rangle}$ to $\langle \{X : \varphi\}, \omega \to \varphi \rangle$ gives us $\langle \{X : \varphi\}, \varphi' \to \varphi \rangle$ and this second one is obtained applying the type substitution $(\varphi' \mapsto \omega)$ to the former one. The variable environment of X and Z in this derivation with $\{b \# Z\}$ is $\{X : \varphi\}$. For the term in the right-hand side, we have the same variable environment, as we can see in the type derivation below.

$$\dfrac{\dfrac{\dfrac{\dfrac{X : \varphi, b : \omega, a : \omega \vdash X : \varphi \ (\mathcal{T}_{\mathbf{X}})}{X : \varphi, b : \omega \vdash [a]X : [\omega]\varphi} \ (\mathcal{T}_{[\mathbf{a}]}) \qquad X : \varphi, b : \omega \vdash Z : \omega \ (\mathcal{T}_{\cap})}{X : \varphi, b : \omega \vdash \mathsf{Sub}([a]X, Z) : \varphi} \ (\mathcal{T}_{\mathsf{f}})}{X : \varphi \vdash [b]\mathsf{Sub}([a]X, Z) : [\omega]\varphi} \ (\mathcal{T}_{[\mathbf{a}]})}{X : \varphi \vdash \mathsf{Lam}[b]\mathsf{Sub}([a]X, Z) : \omega \to \varphi} \ (\mathcal{T}_{\mathsf{f}})$$

Definition 5.10. Let $R = \Phi \Vdash \nabla \vdash l \to r : \rho$ be a typed rewrite rule. A **nominal typed rewrite step** $\Gamma \Vdash \Delta \vdash s \to^{R}_{\tau} t$ holds whenever $(\Phi \Vdash \nabla \vdash l : \rho) \stackrel{?}{\approx}_{\alpha} (\Gamma \Vdash \Delta \vdash s|_p) = (\mathcal{C}, \theta, \pi)$, for some $p \in Pos(s)$, and $\Delta \vdash t \approx_{\alpha} s[p \leftarrow {}^{\pi}r\theta]$.

Table 4: Λ_x

Symbols	**Type Declarations**	**Arity**
$\mathsf{App}:$	$(\varphi_1 \to \varphi_2) \to \varphi_1 \to \varphi_2$	2
$\mathsf{Lam}:$	$([\varphi_1]\varphi_2) \to \varphi_1 \to \varphi_2$	1
$\mathsf{Sub}:$	$([\varphi_1]\varphi_2) \to \varphi_1 \to \varphi_2$	2

Rules:

$$X:\varphi \Vdash \varnothing \vdash \mathsf{App}(\mathsf{Lam}\,[a]X, Y) \longrightarrow \mathsf{Sub}([a]X, Y):\varphi$$
$$X:\varphi \Vdash a\#X \vdash \mathsf{Sub}([a]X, Y) \longrightarrow X:\varphi$$
$$Y:\varphi \Vdash \varnothing \vdash \mathsf{Sub}([a]a, Y) \longrightarrow Y:\varphi$$
$$X:\varphi_1 \to \varphi_2, Y:\varphi_1 \Vdash \varnothing \vdash \mathsf{Sub}([a]\mathsf{App}(X, Y), Z) \longrightarrow \mathsf{App}(\mathsf{Sub}([a]X, Z), \mathsf{Sub}([a]Y, Z)):\varphi_2$$
$$X:\varphi \Vdash b\#Z \vdash \mathsf{Sub}([a]\mathsf{Lam}\,[b]X, Z) \longrightarrow \mathsf{Lam}\,[b]\mathsf{Sub}([a]X, Z):\omega \to \varphi$$

Example 5.11. Take the first rule of the system in Table 4, which is typed and represents the β-reduction in the context of explicit substitutions. The typed matching problem

$$(X{:}\varphi \Vdash \varnothing \vdash \mathsf{App}(\mathsf{Lam}[a]X, Y))^{?}{\approx_\alpha}\ (\varnothing \Vdash \varnothing \vdash \mathsf{App}(\mathsf{Lam}[b]\mathsf{App}(b, b), \mathsf{Lam}[b]b))$$

has a solution $S = ((\varphi \mapsto (\varphi' \to \varphi')), (a\,b), [X \mapsto \mathsf{App}(b, b)] \circ [Y \mapsto \mathsf{Lam}[b]b])$. Look at the type derivation of the left-hand side of the rule.

$$(\mathcal{T}_{\mathsf{f}})\ \frac{(\mathcal{T}_{\mathsf{f}})\ \dfrac{(\mathcal{T}_{[\mathbf{a}]})\ \dfrac{X:\varphi, a:\omega \vdash X:\varphi\ (\mathcal{T}_{\mathbf{X}})}{X:\varphi \vdash [a]X:[\omega]\varphi}}{X:\varphi \vdash \mathsf{Lam}[a]X:\omega \to \varphi} \qquad X:\varphi \vdash Y:\omega\ (\mathcal{T}_{\cap})}{X:\varphi \vdash \mathsf{App}(\mathsf{Lam}[a]X, Y):\varphi}$$

To verify the solution S, it is necessary to check that the permutation and the term substitution satisfy the matching for the terms and freshness contexts, and to see if typability in the instantiated variable environments continues working. The variable environment of Y is trivial because the type is ω. For X, one has that $\varnothing \vdash \mathsf{Lam}[a]a : \varphi' \to \varphi'$.

By Lemma 5.6, $\varnothing \vdash \mathsf{App}(\mathsf{Lam}[b]\mathsf{App}(b, b), \mathsf{Lam}[b]b) : \varphi' \to \varphi'$ is derivable. Indeed, we have the derivations below of the "self-application", of the

"identity" and of the term considered above. Consider $\sigma = \varphi' \to \varphi'$.

$$\mathcal{D}' = \dfrac{(\mathcal{T}_{[\mathbf{a}]})\ \dfrac{(\mathcal{T}_{\mathsf{f}})\ \dfrac{b : (\sigma \to \sigma) \cap \sigma \vdash b : \sigma \to \sigma\ (\mathcal{T}_{\mathbf{a}}) \qquad b : (\sigma \to \sigma) \cap \sigma \vdash b : \sigma\ (\mathcal{T}_{\mathbf{a}})}{b : (\sigma \to \sigma) \cap \sigma \vdash \mathsf{App}(b, b) : \sigma}}{\varnothing \vdash [b]\mathsf{App}(b, b) : [(\sigma \to \sigma) \cap \sigma]\sigma}}{(\mathcal{T}_{\mathsf{f}})\ \dfrac{}{\varnothing \vdash \mathsf{Lam}[b]\mathsf{App}(b, b) : ((\sigma \to \sigma) \cap \sigma) \to \sigma}}$$

$$\mathcal{D}'' = \dfrac{(\mathcal{T}_{\mathsf{f}})\ \dfrac{(\mathcal{T}_{[\mathbf{a}]})\ \dfrac{b : \sigma \vdash b : \sigma\ (\mathcal{T}_{\mathbf{a}})}{\varnothing \vdash [b]b : [\sigma]\sigma} \qquad \dfrac{\dfrac{b : \varphi \vdash b : \varphi\ (\mathcal{T}_{\mathbf{a}})}{\varnothing \vdash [b]b : [\varphi]\varphi}\ (\mathcal{T}_{[\mathbf{a}]})}{\varnothing \vdash \mathsf{Lam}[b]b : \sigma}\ (\mathcal{T}_{\mathsf{f}})}{\varnothing \vdash \mathsf{Lam}[b]b : (\sigma \to \sigma) \cap \sigma}}{(\mathcal{T}_{\cap})}$$

$$(\mathcal{T}_{\mathsf{f}})\ \dfrac{\dfrac{\mathcal{D}'}{\varnothing \vdash \mathsf{Lam}[b]\mathsf{App}(b, b) : ((\sigma \to \sigma) \cap \sigma) \to \sigma} \qquad \dfrac{\mathcal{D}''}{\varnothing \vdash \mathsf{Lam}[b]b : (\sigma \to \sigma) \cap \sigma}}{\varnothing \vdash \mathsf{App}(\mathsf{Lam}[b]\mathsf{App}(b, b), \mathsf{Lam}[b]b) : \varphi' \to \varphi'}$$

5.1. Subject Reduction

Now the Subject Reduction theorem is presented, i.e., the result that proves that typed rewrite steps using typed rewrite rules preserve types under the condition of uniformity of such rules. Notice that uniformity (Definition 5.12) is crucial to have preservation of typings as we can see in the following example. This condition was inspired by Fairweather and Fernández (2018).

Definition 5.12 (Uniformity)**.** We call a nominal typed rewrite rule R **uniform** when, if $\Gamma \Vdash \Delta \vdash s \to^{R}_{\tau} t$ and $\Delta, \Delta' \vdash a\#s$ for some Δ', then $\Delta, \Delta' \vdash a\#t$.

Example 5.13. The rule $R = (X : \varphi \Vdash \varnothing \vdash X \longrightarrow (a\,b) \cdot X : \varphi)$ is typed but it is not uniform and does not preserve typings. Observe that the only variable environment for the left-hand and right-hand side is $\{X : \varphi\}$ and that the matching problem

$$(X : \varphi \Vdash \varnothing \vdash X : \varphi) \stackrel{?}{\approx}_{\alpha} (a : \varphi \Vdash \varnothing \vdash a : \varphi)$$

has a solution $(Id, [X \mapsto a], id)$, but $\{a : \varphi\}$ is not enough to type b, the resulting term of the typed rewrite step. This occurs because non-uniform rules can introduce atoms that are fresh in the left-hand side into the right-hand side.

Theorem 5.14 (Subject Reduction)**.** *Given a uniform typed rewrite rule* $R = \Phi, \nabla \vdash l \to r : \sigma$*, if* $\Gamma \vdash s : \gamma$ *and* $\Gamma \Vdash \Delta \vdash s \to^R_\tau t$*, then* $\Gamma \vdash t : \gamma$*.*

Proof. By the Subterm Lemma, there is a judgement $\Gamma' \vdash s|_p : \gamma'$ whose derivation is part of the derivation of $\Gamma \vdash s : \gamma$ and p is the position where the redex occurs. Indeed, this lemma is considered when $s|_p$ is not inside some s' with type ω in such derivation. In this case, we can always assign the type ω for the corresponding subterm t' of t.

Since $\langle \Phi, \sigma \rangle$ is a typing of l, by Lemma 5.6, $\Gamma' \vdash {}^\pi l\theta : \mathcal{C}(\sigma)$, for a solution $(\mathcal{C}, \theta, \pi)$ for the matching problem $(\Phi \Vdash \nabla \vdash l : \sigma) \stackrel{?}{\approx}_\alpha (\Gamma' \Vdash \Delta \vdash s|_p)$. Insofar as $\Delta \vdash {}^\pi l\theta \approx_\alpha s|_p$, it holds also that $\Gamma' \vdash {}^\pi l\theta : \gamma'$. By Lemma 5.7, there exists Γ^* that differs from Γ' only in the unknowns which are in l and such that $\Gamma^* \vdash {}^\pi l : \gamma'$.

By the principality of $\langle \Phi, \sigma \rangle$ for l, $\langle {}^\pi\Phi, \sigma \rangle$ is principal for ${}^\pi l$, by Lemma 4.11 (meta-level equivariance). So, there exists $\mathcal{C}'$ such that $\langle \Gamma^*, \gamma' \rangle = \mathcal{C}'(\langle {}^\pi\Phi, \sigma \rangle)$ and $(\mathcal{C}', \theta, \pi)$ is also a solution for the typed matching problem $(\Phi \Vdash \nabla \vdash l : \sigma) \stackrel{?}{\approx}_\alpha (\Gamma^* \Vdash \Delta \vdash s|_p)$. To conclude the proof, we need to show that $\Gamma' \vdash {}^\pi r\theta : \gamma'$, which can be achieved by using the Matching Lemma (Lemma 5.6) if we are able to prove that $(\mathcal{C}', \pi, \theta)$ is a solution of the typed matching problem

$$(\Phi \Vdash \nabla \vdash r : \sigma) \stackrel{?}{\approx}_\alpha (\Gamma^* \Vdash \Delta \vdash {}^\pi r\theta)$$

because the additional unknowns that are annotated in Γ^* do not occur in ${}^\pi r\theta$.

In fact, one must demonstrate that, for all $X \in unkn(r)$, ${}^{\pi'^{-1}}\Gamma^* \bowtie \mathcal{C}'(\Phi') \vdash X\theta : \mathcal{C}'(\Phi)_X$ is derivable, as described in Definition 5.3, for π' in ${}^\pi r$. The necessary type annotations in ${}^{\pi'^{-1}}\Gamma^* \bowtie \mathcal{C}'(\Phi')$ to type the term $X\theta$ are of the unknowns and free atoms introduced by θ, unless they are typed with ω. The different permutations do not change the type annotations of unknowns. So we will concentrate on the free atoms of $X\theta$.

Take a a free atom in $X\theta$. Consider $\pi_1, \ldots, \pi_k$ the permutations that accompany X in ${}^\pi l$. Since ${}^{\pi_i^{-1}}\Gamma^* \bowtie \mathcal{C}'(\Phi') \vdash X\theta : \mathcal{C}'(\Phi)_X$ is derivable for all $i = 1, \ldots, k$, we know that $a : \rho \in \mathcal{C}'(\Phi')$ or $a : \rho \in {}^{\pi_i^{-1}}\Gamma^*$ for some $\rho \neq \omega$. In the first case, it is all right because $a : \rho$ would be also in ${}^{\pi'^{-1}}\Gamma^* \bowtie \mathcal{C}'(\Phi')$. In the second case, $\pi_i(a) : \rho \in \Gamma^*$ for all $i = 1, \ldots, k$. If $\pi'(a) = \pi_i(a)$ for some i, then it is done, because $a : \rho$ would be in ${}^{\pi'^{-1}}\Gamma^*$. Otherwise, by the uniformity of the rule, $\pi'(a)$ would be abstracted over $\pi' \cdot X$. However, in this case, $a : \rho$ should be initially in $\mathcal{C}'(\Phi')$, as already considered. □

6. Conclusions and Future Work

In this paper, we extended the study on types for nominal terms developing an intersection type system with the subject reduction property. We have chosen to present an adapted Essential System, presented first in Bakel (1993) in the context of the λ-calculus. Here, the syntax of types is extended with user defined type constructors and an abstraction type constructor. These extra types required modifications on the construction of the type ordering as well as of the type operations of lifting, substitution and expansion. A specialised type inference system for nominal terms was also built. The notion of typability differs from Fairweather (2014) because there is no restriction over the permutations of a suspension and no condition over type derivations such as the "diamond property".

The type system satisfies the expected properties regarding compatibility of type operations with respect to the type ordering, equivariance of typings under meta- and object-level action of permutations, and preservation of typings for α-equivalent terms. With respect to rewriting, the notion of matching needed to be replaced by a typed matching as well as the notion of rewriting. The constraints that must be imposed are due to the possibly capturing nature of our substitutions. With the adaptations on the definitions and the condition of uniformity on the rewrite rules, we were able to prove preservation of typing under typed rewrite steps, i.e., the subject reduction property.

Future work:. Intersection type systems are a useful tool to study normalisation properties of terms. Unlike the λ-calculus or explicit substitutions calculi, rewriting rules in our nominal setting are not predetermined, which means that suitable general conditions have to be devised in order to ensure normalisation for some user-defined rewriting system. Similar work has been done for term rewriting systems in Bakel and Fernández (1997).

The notion of typed rewriting can also be extended to the closed rewriting relation, as done for the polymorphic system in Fairweather (2014). There, it is shown that, although the typed rewriting relation is more expressive, the typed closed rewriting relation is more efficient and most of the systems of interest can be modelled by the latter approach.

Additionally, criteria to guarantee the principal pair property have to be investigated. We do believe that such feature can be achieved for some restricted version of our type system, since the one presented in Bakel et al.

(1996), which combines term rewriting systems with the λ-calculus, has the principal pair property for an intersection type system. That system is close to ours in the sense that it possesses function symbols with type declarations and λ-abstractions.

Acknowledgements

This work was partially supported by the Brazilian Coordination for the Improvement of Higher Education Personnel CAPES within grant PVE 0500/2013 and the Federal District Research Support Foundation FAPDF within grant DE 0193001369/2016. The first author was partially supported by a research grant from the Brazilian National Council for Scientific and Technological Development CNPq and the third by a scholarship from CAPES.

7. References

Ayala-Rincón, M., Kamareddine, F., 2001. Unification via the λs_e-Style of Explicit Substitution. Logic Journal of the IGPL 9 (4), 489–523.

Bakel, S. van., 1993. Essential intersection type assignment. In: Foundations of Software Technology and Theoretical Computer Science, 13th Conference, Bombay, India, December 15-17, 1993, Proceedings. pp. 13–23.
URL http://dx.doi.org/10.1007/3-540-57529-4_40

Bakel, S. van., 1995. Intersection Type Assignment Systems. Theoretical Computer Science 151 (2), 385–435.
URL http://dx.doi.org/10.1016/0304-3975(95)00073-6

Bakel, S. van., 2011. Strict intersection types for the lambda calculus. ACM Computer Surveys 43 (3), 20.
URL http://doi.acm.org/10.1145/1922649.1922657

Bakel, S. van., Barbanera, F., Fernández, M., 1996. Rewrite Systems with Abstraction and beta-Rule: Types, Approximants and Normalization. In: Programming Languages and Systems - ESOP'96, 6th European Symposium on Programming, Proceedings. Vol. 1058 of Lecture Notes in Computer Science. Springer Verlag, pp. 387–403.
URL http://dx.doi.org/10.1007/3-540-61055-3_50

Bakel, S. van., Fernández, M., 1997. Normalization results for typeable rewrite systems. Information and Computation 133 (2), 73–116.
URL http://dx.doi.org/10.1006/inco.1996.2617

Barendregt, H., Coppo, M., Dezani-Ciancaglini, M., 1983. A Filter Lambda Model and the Completeness of Type Assignment. J. of Symbolic Logic 48 (4), 931–940.
URL http://dx.doi.org/10.2307/2273659

Barendregt, H. P., Dekkers, W., Statman, R., 2013. Lambda Calculus with Types. Perspectives in logic. Cambridge University Press.
URL http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types

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

Castagna, G., Dezani-Ciancaglini, M., Giachino, E., Padovani, L., 2009. Foundations of session types. In: Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming. pp. 219–230.

Cheney, J., 2009. A Simple Nominal Type Theory. Electronic Notes in Theoretical Computer Science 228, 37–52.
URL http://dx.doi.org/10.1016/j.entcs.2008.12.115

Cheney, J., 2012. A Dependent Nominal Type Theory. Logical Methods in Computer Science 8 (1).
URL http://dx.doi.org/10.2168/LMCS-8(1:8)2012

Coppo, M., Dezani-Ciancaglini, M., 1978. A New Type Assignment for λ-terms. Archiv für mathematische Logik und Grundlagenforschung 19 (1), 139–156.

Coppo, M., Dezani-Ciancaglini, M., Venneri, B., 1981. Functional Characters of Solvable Terms. Mathematical Logic Quarterly 27 (26), 45–58.

Dowek, G., Hardin, T., Kirchner, C., 2000. Higher-order Unification via Explicit Substitutions. Information and Computation 157 (1/2), 183–235.

Dunfield, J., 2014. Elaborating intersection and union types. J. of Functional Programming 24 (2-3), 133–165.

Espírito Santo, J., Ivetic, J., Likavec, S., 2012. Characterising strongly normalising intuitionistic terms. Fundamenta Informaticae 121 (1-4), 83–120.

Fairweather, E., 2014. Type Systems for Nominal Terms. Ph.D. thesis, King's College London.

Fairweather, E., Fernández, M., 2018. Typed nominal rewriting. ACM Trans. Comput. Log. 19 (1), 6:1–6:46.
URL http://doi.acm.org/10.1145/3161558

Fairweather, E., Fernández, M., Gabbay, M. J., 2011. Principal Types for Nominal Theories. In: International Symposium on Fundamentals of Computation Theory FCT. Vol. 6914 of Lecture Notes in Computer Science. Springer Verlag, pp. 160–172.

Fairweather, E., Fernández, M., Szasz, N., Tasistro, A., 2015. Dependent Types for Nominal Terms with Atom Substitutions. In: 13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland. pp. 180–195.
URL http://dx.doi.org/10.4230/LIPIcs.TLCA.2015.180

Fernández, M., Gabbay, M. J., 2006. Curry-Style Types for Nominal Terms. In: Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers. pp. 125–139.
URL http://dx.doi.org/10.1007/978-3-540-74464-1_9

Fernández, M., Gabbay, M. J., 2007. Nominal rewriting. Information and Computation 205 (6), 917–965.

Gabbay, M. J., Pitts, A. M., 1999. A New Approach to Abstract Syntax Involving Binders. In: 14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999. pp. 214–224.
URL http://dx.doi.org/10.1109/LICS.1999.782617

Ghilezan, S., Ivetic, J., Lescanne, P., Likavec, S., 2011. Intersection types for the resource control lambda calculi. In: Antonio Cerone, P. P. (Ed.), the 8th International Colloquium on Theoretical Aspects of Computing

(ICTAC). Vol. 6916 of Lecture Notes in Computer Science. Springer Verlag, pp. 116–134.

Girard, J.-Y., Lafont, Y., Taylor, P., 1989. Proofs and Types. Vol. 7 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Hindley, J. R., 2002. Basic Simple Type Theory. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Huet, G. P., 1975. A Unification Algorithm for Typed $\vec{\lambda}$-Calculus. Theoretical Computer Science 1, 27–57.

Kamareddine, F., Nour, K., 2007. A Completeness Result for a Realisability Semantics for an Intersection Type System. Annals of Pure and Applied Logic 146 (2-3), 180–198.
URL http://dx.doi.org/10.1016/j.apal.2007.02.001

Kesner, D., Ventura, D., 2014. Quantitative types for the linear substitution calculus. In: Díaz, J., Lanese, I., Sangiorgi, D. (Eds.), the 8th International Conference on Theoretical Computer Science (TCS). Vol. 8705 of Lecture Notes in Computer Science. Springer Verlag, pp. 296–310.

Kesner, D., Ventura, D., 2017. A resource aware semantics for a focused intuitionistic calculus. Mathematical Structures in Computer Science, 134.
URL http://dx.doi.org/10.1017/S0960129517000111

Kfoury, A., Wells, J. B., 2004. Principality and type inference for intersection types using expansion variables. Theoretical Computer Science 311 (1-3), 1–70.

Kumar, R., Norrish, M., 2010. (Nominal) Unification by Recursive Descent with Triangular Substitutions. In: Interactive Theorem Proving, First International Conference, ITP 2010. Vol. 6172 of Lecture Notes in Computer Science. Springer Verlag, pp. 51–66.
URL http://dx.doi.org/10.1007/978-3-642-14052-5_6

Lengrand, S., Lescanne, P., Dougherty, D. J., Dezani-Ciancaglini, M., Bakel, S. van., 2004. Intersection Types for Explicit Substitutions. Information and Computation 189 (1), 17–42.
URL http://dx.doi.org/10.1016/j.ic.2003.09.004

Padovani, L., 2012. On projecting processes into session types. Mathematical Structures in Computer Science 22 (2), 237–289.

Piccolo, M., 2012. Strong normalization in the π-calculus with intersection and union types. Fundamenta Informaticae 121 (1-4), 227–252.

Pitts, A. M., 2003. Nominal Logic, a First Order Theory of Names and Binding. Information and Computation 186 (2), 165–193.
URL http://dx.doi.org/10.1016/S0890-5401(03)00138-X

Pitts, A. M., Matthiesen, J., Derikx, J., 2015. A Dependent Type Theory with Abstractable Names. Electronic Notes in Theoretical Computer Science 312, 19–50.
URL http://dx.doi.org/10.1016/j.entcs.2015.04.003

Stehr, M.-O., 2000. CINNI - A Generic Calculus of Explicit Substitutions and its Application to λ- ς- and π-Calculi. Electronic Notes in Theoretical Computer Science 36, 70–92, the 3rd Int. Workshop on Rewriting Logic and its Applications.
URL http://www.sciencedirect.com/science/article/pii/S1571066105801252

Urban, C., Pitts, A. M., Gabbay, M. J., 2004. Nominal Unification. Theoretical Computer Science 323 (1-3), 473–497.
URL http://dx.doi.org/10.1016/j.tcs.2004.06.016

Ventura, D. L., Kamareddine, F., Ayala-Rincón, M., 2015. Explicit Substitution Calculi with de Bruijn Indices and Intersection Type Systems. Logic Journal of the IGPL 23 (2), 295–340.
URL http://dx.doi.org/10.1093/jigpal/jzu044

Wells, J. B., 2002. The essence of principal typings. In: Widmayer, P., Ruiz, F. T., Bueno, R. M., Hennessy, M., Eidenbenz, S., Conejo, R. (Eds.), the 29th International Colloquium on Automata, Languages and Programming (ICALP). Vol. 2380 of Lecture Notes in Computer Science. Springer Verlag, pp. 913–925.