# Polymorphic Intersection Type Assignment for Rewrite Systems with Abstraction and $\beta$-rule[*]

Steffen van Bakel[1], Franco Barbanera [2], and Maribel Fernández [3]

[1] Department of Computing, Imperial College,
180 Queen's Gate, London SW7 2BZ, U.K.

[2] Dipartimento di Matematica e Informatica, Università degli Studi di Catania,
Viale A. Doria 6, 95125 Catania, Italia.

[3] DI - LIENS, École Normale Supérieure / CNRS,
45, rue d'Ulm, 75005 Paris, France.

svb@doc.ic.ac.uk, barba@dipmat.unict.it, maribel@di.ens.fr

**Abstract**

We define two type assignment systems for first-order rewriting extended with application, $\lambda$-abstraction, and $\beta$-reduction, using a combination of intersection types and second-order polymorphic types. The first system is the general one, for which we prove subject reduction, and strong normalisation of typeable terms. The second is a decidable subsystem of the first, by restricting to rank 2 (intersection and quantified) types. For this system we define, using an extended notion of unification, a notion of principal typing which is more general than ML's principal type property, since also the types for the free variables of terms are inferred.

## Introduction

Since the first investigations on combinations of Lambda Calculus (LC) and Term Rewriting Systems (TRS) [13, 19, 14, 26], this topic has drawn attention from the theoretical computer science community. At first, the area of programming language design was consider to be the typical field on which the theoretical results for combinations of the two computational paradigms could better exploit their potentialities. Later on, the evolution of interactive proof development tools with inductive types, and theorem provers in general, disclosed a number of possible applications.

Apart from the practical outcome, most of the theoretical investigations in this particular field have shown that type disciplines provide an optimal environment in which rewrite rules and $\beta$-reduction can interact without loss of their useful properties [14, 15, 26, 9, 10, 7]. Type disciplines come in two main flavours: *explicitly typed* (also called *à la Church*) and *type inference systems* (*à la Curry*). Systems based on the latter sort of type discipline are of great interest from the point of view of programming language design. In fact, they save the programmer from specifying a type for each variable (i.e. no type annotation is required). Most of the results about combinations of LC and TRS, however, concern systems which are explicitly typed.

In the context of the LC alone, type inference disciplines have been widely studied, in particular with intersection types, and some work has also been done for TRS alone, more precisely, for *Curryfied* TRS ($\mathcal{C}$TRS) [8] which are first-order TRS with application, that correspond to the TRS underlying the programming language Clean [34]. The interactions between LC and TRS for type systems à la Curry, instead, has not been extensively investigated. They

[*] Partially supported by NATO Collaborative Research Grant CRG 970285 'Extended Rewriting and Types'.

were first studied in [7], where $\mathcal{C}$TRS extended with $\lambda$-abstraction and $\beta$-reduction were defined, together with a notion of intersection type assignment for both the LC and the TRS fragments. In this paper we carry on this study by taking *explicit polymorphism* into account, namely the possibility, from the programming language point of view, of using the same program with arguments of different types.

We take into account also another important feature of type systems for programming languages: the notion of *principal type*, that is, a type from which all the other types of a term can be derived. As a matter of fact we consider an even stronger property than principal types: *the principal typing property*. This means that any *typing judgement* for a term can be obtained from the principal one, that is, not only the type but also the basis (containing the type assumptions for the free variables) is obtained. The pragmatic value of this property is demonstrated in [25], where it is shown that, unlike principal types, principal typings provide support for separate compilation, incremental type inference, and for accurate type error messages.

The type system of ML is polymorphic and has principal types, but its polymorphism is limited (some programs that arise naturally cannot be typed), and it does *not* have principal typings (see [17, 25]). System F [23] provides a much more general notion of polymorphism, but lacks principal types, and type inference is undecidable in general (although it is decidable for some subsystems, in particular if we consider types of rank 2 [27]). Intersection type systems [12] are somewhere in the middle with respect to polymorphism, and have principal typings. But type assignment is again undecidable; decidability is recovered if we restrict ourselves to intersection types of finite rank [29].

In [24], a system for LC combining intersection types and System F was presented, which has principal typings (see [32]). In this paper we extend the approach of that system to a combination of LC and $\mathcal{C}$TRS. In other words, we extend the type system of [7] further, adding '$\forall$' as an extra type-constructor (i.e. explicit polymorphism). Although extending the set of types by adding '$\forall$' does not extend the expressivity of the system in terms of typeable terms, the set of assignable types increases, and types can better express the behaviour of terms (see [16]). The resulting system has the expected properties: Subject Reduction, and Strong Normalization when the rewrite rules use a limited form of recursion (inspired by the General Schema of Jouannaud and Okada [26]). The proof of the latter follows the method of Tait-Girard's reducibility candidates, extended in order to take the presence of (higher-order) algebraic rewriting into account.

In view of the above results, two questions arise naturally:

*i*) Is the Rank 2 combination of System F and the Intersection System also decidable?

*ii*) Does it have the principal typing property?

We answer these questions in the affirmative. The restriction to types of rank 2 of the combined system of polymorphic and intersection types is decidable. This restricted system can be seen as a combination of the systems considered in [6] and [27]. The combination is two-fold: not only the type systems of those two papers are combined (resp. intersection and polymorphic types of rank 2), but also their calculi are combined (resp. $\mathcal{C}$TRS and LC). In our Rank 2 system each typeable term has a principal typing. This is the case also in the Rank 2 intersection system of [6], but not in the Rank 2 polymorphic system of [27]. For the latter, a type inference algorithm of the same complexity of that of ML was given in [28], where the problems that occur due to the lack of principal types are discussed in detail. Our Rank 2 system generalizes also Jim's system $P_2$ [25], which is a combination of ML-types and Rank 2 intersection types. Having Rank 2 quantified types in the system allows us to type for instance the constant `runST` used in [31], which cannot be typed in $P_2$.

This paper is organised as follows: In Section 1 we define TRS with application, $\lambda$-abstraction and $\beta$-reduction (TRS $+\,\beta$), and in Section 2 the type assignment system. Section 3 deals with the strong normalization property for typeable terms, and in Section 4 we prove a confluence result. In Section 5 we present the restriction of the general type assignment system to Rank 2; finally, Section 6 contains the conclusions.

## 1 Term Rewriting Systems with $\beta$-reduction rule

In this section, we will present a combination of untyped Lambda Calculus with untyped Algebraic Rewriting, obtained by extending first-order TRS with notions of application and abstraction, and a $\beta$-reduction rule. We can look at such calculi also as extensions of the Curryfied Term Rewriting Systems (CuTRS) considered in [3, 8], by adding $\lambda$-abstraction and a $\beta$-reduction rule. We assume the reader to be familiar with LC [11] and refer to [30, 18] for rewrite systems.

**Definition 1.1** *i*) An *alphabet* or *signature* $\Sigma$ consists of a countable, infinite set $\mathcal{X}$ of variables $x_1, x_2, x_3, \ldots$ (or $x, y, z, x', y', \ldots$), a non-empty set $\mathcal{F}$ of *function symbols* $F$, $G$, $\ldots$, each with a fixed arity, and a special binary operator, called *application* ($Ap$).

*ii*) The set $T(\mathcal{F},\mathcal{X})$ of *terms*, ranged over by $t$, is defined by:

$$t ::= x \mid F(t_1,\ldots,t_n) \mid Ap(t_1,t_2) \mid \lambda x.t$$

We will consider terms modulo $\alpha$-conversion.
A *context* is a term with a hole, and it is denoted as usual by $\mathrm{C}[\,]$.

*iii*) *a*) A *neutral* term is a term not of the form $\lambda x.t$.

*b*) A *lambda term* is a term not containing function symbols.

*c*) An *algebraic term* is a term containing neither $\lambda$ nor $Ap$.

The set of *free variables* of a term $t$ is defined as usual, and denoted by $FV(t)$.

A term-substitution is an operation on terms where terms variables are replaced by terms; as usual for term-substitution in systems with abstraction, we will consider terms modulo $\alpha$-conversion to avoid name clashes. To denote a term-substitution, we use capital characters like 'R', instead of Greek characters like '$\sigma$', which will be used to denote types. Sometimes we use the notation $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. We write $t^{\mathrm{R}}$ for the result of applying the term-substitution R to $t$.

Reductions on $T(\mathcal{F},\mathcal{X})$ are defined through rewrite rules together with a $\beta$-reduction rule.

**Definition 1.2** (Reduction) *i*) A *rewrite rule* is a pair $(l,r)$ of terms. Often, a rewrite rule will get a name, e.g. $\mathbf{r}$, and we write $l \rightarrow_{\mathbf{r}} r$. Three conditions are imposed: $l \notin \mathcal{X}$, $l$ is an algebraic term, and $FV(r) \subseteq FV(l)$.

*ii*) A rewrite rule $l \rightarrow r$ determines a set of *rewrites* $l^{\mathrm{R}} \rightarrow r^{\mathrm{R}}$ for all term-substitutions R. The left hand side $l^{\mathrm{R}}$ is called a *redex*, the right hand side $r^{\mathrm{R}}$ its *contractum*. Likewise, for any $t$ and $u$, the usual notion of $\beta$-reduction $Ap(\lambda x.t,u) \rightarrow_{\beta} t^{\{x \mapsto u\}}$ is also a rewrite; $Ap(\lambda x.t,u)$ is called a redex, and $t^{\{x \mapsto u\}}$ its contractum.

*iii*) A redex $t$ may be substituted by its contractum $t'$ inside a context $\mathrm{C}[\,]$; this gives rise to *rewrite steps* $\mathrm{C}[t] \rightarrow \mathrm{C}[t']$. Concatenating rewrite steps we have *rewrite sequences* $t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots$. If $t_0 \rightarrow \cdots \rightarrow t_n$ ($n \geq 0$) we also write $t_0 \rightarrow^* t_n$, and $t_0 \rightarrow^+ t_n$ if $t_0 \rightarrow^* t_n$ in one step or more.

**Definition 1.3** A *Term Rewriting System with $\beta$-reduction rule* (TRS$+\beta$) is defined by a pair $(\Sigma, \mathbf{R})$ of an alphabet $\Sigma$ and a set $\mathbf{R}$ of rewrite rules.

Note that in contrast with $\mathcal{C}$TRS, the rewrite rules considered in this paper may contain $\lambda$-abstractions in the right-hand sides.

We take the view that in a rewrite rule a certain symbol is defined.

**Definition 1.4** In a rewrite rule $F(t_1,\ldots,t_n) \to_{\mathbf{r}} r$, $F$ is called *the defined symbol* of $\mathbf{r}$, and $\mathbf{r}$ is said to *define* $F$. $F$ is *a defined symbol*, if there is a rewrite rule that defines $F$, and $Q \in \mathcal{F}$ is called a *constructor* if $Q$ is not a defined symbol.

Notice that '$Ap$' cannot be a defined symbol since it cannot appear in the left-hand side of a rewrite rule.

**Definition 1.5** Let $(\Sigma,\mathbf{R})$ be a TRS $+\beta$.

*i*) A term is in *normal form* if it contains no redex.
*ii*) A term $t$ is *strongly normalisable* if all the rewrite sequences starting with $t$ are finite, and we will write $\mathcal{SN}(t)$ if $t$ is strongly normalisable.
*iii*) $(\Sigma,\mathbf{R})$ is *strongly normalising* if every term is.
*iv*) $(\Sigma,\mathbf{R})$ is *confluent* if for all $t$ such that $t \to^* u$ and $t \to^* v$, there exists $s$ such that $u \to^* s$ and $v \to^* s$.

*Example 1.6* The following is a set of rewrite rules that defines the functions append and map on lists and establishes the associativity of append. The function symbols nil and cons are constructors.

$$\begin{array}{ll}
\mathsf{append}\,(\mathsf{nil},l) & \to l \\
\mathsf{append}\,(\mathsf{cons}\,(x,l),l') & \to \mathsf{cons}\,(x,\mathsf{append}\,(l,l')) \\
\mathsf{append}\,(\mathsf{append}\,(l,l'),l'') & \to \mathsf{append}\,(l,\mathsf{append}\,(l',l'')) \\
\mathsf{map}\,(f,\mathsf{nil}) & \to \mathsf{nil} \\
\mathsf{map}\,(f,\mathsf{cons}\,(y,l)) & \to \mathsf{cons}\,(Ap(f,y),\mathsf{map}\,(f,l))
\end{array}$$

Since variables in TRS $+\beta$ may be substituted by $\lambda$-expressions, we obtain the usual functional programming paradigm, extended with definitions of operators and data structures. Notice, however, that we obtain more: in functional programs, the set $\mathcal{F}$ is divided into *function symbols* and *constructors*, and, in rewrite rules, function symbols are not allowed to appear in 'constructor position' and vice-versa. This does not hold for TRS $+\beta$: the symbol 'append' appears in the third rule in both function and constructor position.

## 2 A Polymorphic Intersection System for TRS $+\beta$

In this section we define a type assignment system for TRS+$\beta$, that can be seen as an extension (by adding '$\forall$') of the intersection system presented in [7]. We use polymorphic strict intersection types, defined over a set of type-variables and sorts (type constants), using arrow, intersection and universal quantifiers. We assume the reader to be familiar with intersection type assignment systems, and refer to [12, 2, 5] for more details.

Several systems can be considered as fragments of our type assignment system:

*i*) The system of [7]: $\forall$-free fragment
*ii*) The system of [8]: $\lambda$-free, $\forall$-free fragment
*iii*) The system of [5]: sort-free, $\forall$-free, LC-fragment
*iv*) The type assignment version of System F [23] : intersection-free LC-fragment
*v*) The system of [32]: sort-free LC-fragment

Indeed, the latter system is not a proper fragment, since we use *strict* intersection types (i.e. an intersection type cannot be the right-hand side of an arrow type). However this is not an actual difference, since the use of strict intersection types, while simplifying the typing procedures, does not affect the typing power. Any term typeable in the full intersection type discipline can be given a strict type and vice-versa [2].

For what concerns System F, its type assignment version can be seen as a fragment of our system, but our system is not a pure intersection-extension: we cannot quantify intersection types. Again, this is not a real problem: for any universally quantified intersection type $\forall\alpha.\sigma_1{\cap}\sigma_2$ we have an equivalent type of the form $(\forall\alpha.\sigma_1){\cap}(\forall\alpha.\sigma_2)$.

For LC, a type assignment system that combines the intersection system [12] with System F has been defined in [24] and its principal type property has been studied in [32]. As far as types are concerned, the difference between our system and the latter is that we add constant types, and use strict intersection types.

## 2.1 Types

We use strict intersection types over a set $V = \Phi \uplus \mathcal{A}$ of *type-variables*, and a set $\mathcal{S}$ of *sorts*. For various reasons (definition of operations on types, definition of unification), we will distinguish syntactically between (names of) *free* type-variables (which belong to $\Phi$) and (names of) *bound* type-variables (in $\mathcal{A}$).

**Definition 2.1** (Types) Let $\Phi = \{\varphi_0,\varphi_1,\ldots\}$, $\mathcal{A} = \{\alpha_0,\alpha_1,\ldots\}$, and $\mathcal{S} = \{s_0,s_1,\ldots\}$ all be denumerable sets. $\mathcal{T}_s$, the set of *polymorphic strict types*, and $\mathcal{T}$, the set of *polymorphic strict intersection types*, are defined by:

$$\begin{array}{rcl}\mathcal{T}_s & ::= & \varphi \mid s \mid (\mathcal{T} \to \mathcal{T}_s) \mid \forall\alpha.\mathcal{T}_s[\alpha/\varphi] \\ \mathcal{T} & ::= & (\mathcal{T}_s \cap \cdots \cap \mathcal{T}_s)\end{array}$$

To avoid parentheses in the notation of types, '$\to$' is assumed to associate to the right and, as in logic, '$\cap$' binds stronger than '$\to$', which binds stronger than '$\forall$'; so $\rho{\cap}\mu{\to}\forall\alpha.\gamma{\to}\delta$ stands for $((\rho{\cap}\mu){\to}(\forall\alpha.(\gamma{\to}\delta)))$. Also $\forall\overline{\alpha}.\sigma$ is used as abbreviation for $\forall\alpha_1.\forall\alpha_2\ldots\forall\alpha_n.\sigma$, and we assume that each variable is bound at most once in a type (renaming if necessary). In the meta-language, we denote by $\sigma[\tau/\varphi]$ (resp. $\sigma[\tau/\alpha]$) the substitution of the type-variable $\varphi$ (resp. $\alpha$) by $\tau$ in $\sigma$.

$FV(\sigma)$, the set of *free variables* of a type $\sigma$ is defined as usual (note that by construction, $FV(\sigma) \subseteq \Phi$). A type is called *closed* if it contains no free variables, and *ground* if it contains no variables at all.

**Definition 2.2** (Relations on types) The relation $\leq$ is defined as the least pre-order (i.e. reflexive and transitive relation) on $\mathcal{T}$ such that:

$$\begin{array}{rcll}\forall n \geq 1, \forall 1{\leq}i{\leq}n\ [\sigma_1{\cap}\cdots{\cap}\sigma_n & \leq & \sigma_i] & \\ \forall\alpha.\sigma & \leq & \sigma[\tau/\alpha] & \\ \sigma & \leq & \forall\alpha.\sigma, & \alpha \text{ not in } \sigma \\ \forall n \geq 1, \forall 1{\leq}i{\leq}n\ [\sigma \leq \sigma_i] & \Rightarrow & \sigma \leq \sigma_1{\cap}\cdots{\cap}\sigma_n & \\ \sigma \leq \tau & \Rightarrow & \forall\alpha.\sigma[\alpha/\varphi] \leq \forall\alpha.\tau[\alpha/\varphi] & \\ \rho \leq \sigma \ \&\ \tau \leq \mu & \Rightarrow & \sigma{\to}\tau \leq \rho{\to}\mu & \end{array}$$

The equivalence relation $\sim$ on types is defined by: $\sigma \sim \tau \iff \sigma \leq \tau \leq \sigma$, and we will work with types modulo $\sim$.

Notice that $\mathcal{T}_s$ is a proper subset of $\mathcal{T}$, and that $\sigma{\to}(\tau{\cap}\rho)$ is not a type in $\mathcal{T}$ (it is not strict). To obtain a notion of type assignment that is a true extension of System F, the '$\forall$' type-constructor is allowed to occur on the right of an arrow, so a type like $\sigma{\to}\forall\alpha.\tau$ is well-defined. Also note that we cannot quantify intersection types, but we have equivalent types of the form $(\forall\alpha.\sigma_1){\cap}(\forall\alpha.\sigma_2)$.

## 2.2 Type assignment

Before coming to the definition of type assignment, we introduce the notion of basis.

**Definition 2.3** (STATEMENT AND BASIS) *i*) A *statement* is an expression of the form $t{:}\sigma$, with $\sigma \in \mathcal{T}$ and $t \in T(\mathcal{F},\mathcal{X})$. $t$ is the *subject* and $\sigma$ the *predicate* of $t{:}\sigma$.

*ii*) A *basis* $B$ is a partial mapping from variables to types, represented as set of statements with only distinct variables as subjects.

*iii*) For bases $B_1,\ldots,B_n$, the basis $\Pi\{B_1,\ldots,B_n\}$ is defined by: $x{:}\sigma_1\cap\cdots\cap\sigma_m \in \Pi\{B_1,\ldots,B_n\}$ if and only if $\{x{:}\sigma_1,\ldots,x{:}\sigma_m\}$ is the (non-empty) set of all statements about $x$ that occur in $B_1\cup\cdots\cup B_n$.

*iv*) We extend $\leq$ and $\sim$ to bases by: $B \leq B'$ if and only if for every $x{:}\sigma' \in B'$ there is an $x{:}\sigma \in B$ such that $\sigma \leq \sigma'$, and $B \sim B'$ if and only if $B \leq B' \leq B$.

Notice that if $n = 0$, then $\Pi\{B_1,\ldots,B_n\} = \emptyset$. We will often write $B, x{:}\sigma$ for the basis $\Pi\{B,\{x{:}\sigma\}\}$, when $x$ does not occur in $B$, and write $B\backslash x$ for the basis obtained from $B$ by removing the statement that has $x$ as subject.

One of the main features of type assignment systems, and intersection systems in particular, is to provide *flexibility* of typing. This feature seems in contrast with the type *rigidity* implicitly possessed by function symbols. They have precise arities and a precise functional behaviour, as expressed by the rewriting rules. Developing a type assignment system containing function symbols necessarily implies a sort of mediation, for what concerns algebraic terms, between *flexibility* and *rigidity*. We achieve this by using an *environment* providing a type for each function symbol. This approach, however, is not rigid: from the type provided by the environment we can derive many types to be used for different occurrences of the symbol in a term, all of them 'consistent' with that provided type.

**Definition 2.4** (ENVIRONMENT) An *environment* is a mapping $\mathcal{E} : \mathcal{F} \to \mathcal{T}_s$.

### 2.2.1 Operations on Types

In order to obtain valid instances of the type provided by an environment for a function symbol we will use operations which are standard in type systems with intersection types, suitably modified in order to take into account the presence of universal quantifiers. These operations are: *substitution*, *expansion*, *lifting* and *closure*.

In type systems based on arrow types with type-variables, the operation of *substitution* generates all valid instances of a given type by replacing types for type variables. In presence of intersection types, valid instances could also be the result of replacing (sub)types by the intersection of a number of renamed copies of that (sub)type. This is (roughly) what is performed by the operation of *expansion*. The operation of *lifting*, instead, generates instances of types using the '$\leq$' relation. The last operation we consider, *closure*, is not present in other type systems with intersection types and has been devised to deal in particular with universal quantification.

In the following we shall have to consider the notion of *principal typing* for a term, that is the typing from which all the possible typings for the term can be derived. This can be achieved by means of the above discussed operations. This implies that we shall have to define the above operations not only on types, but also on type derivations (denoted by $B \vdash_{\mathcal{E}} t{:}\sigma$, where $B$ is a basis and $\sigma$ a type). We shall use the same symbols to denote the two versions of the operations, since the context will always clarify any possible ambiguity.

We will show that all operations are sound in the sense that they preserve typeability, and therefore well-defined when extended to derivations. That is, for any operation *op*, if

$B \vdash_{\mathcal{E}} t{:}\sigma$ then $op(B \vdash_{\mathcal{E}} t{:}\sigma)$ is a correct derivation. Of course the variants of the operations on derivations can be composed.

**Substitution.** We will define substitution as usual in first-order logic, but avoid to go out of the set of polymorphic strict intersection types. For example, the replacement of $\varphi$ by $\tau_1\cap\tau_2$ would transform $\sigma\to\varphi$ into $\sigma\to\tau_1\cap\tau_2$, which is not in $\mathcal{T}$. The following definition takes this fact into account.

**Definition 2.5** (Substitution) The *substitution* $(\varphi \mapsto \rho) : \mathcal{T} \to \mathcal{T}$, where $\varphi$ is a type-variable in $\Phi$ and $\rho \in \mathcal{T}_s$, is defined by:

$$\begin{array}{lll}
(\varphi \mapsto \rho)(\varphi) & = \rho \\
(\varphi \mapsto \rho)(\varphi') & = \varphi', \text{ if } \varphi' \neq \varphi \\
(\varphi \mapsto \rho)(s) & = s \\
(\varphi \mapsto \rho)(\alpha) & = \alpha \\
(\varphi \mapsto \rho)(\sigma\to\tau) & = (\varphi \mapsto \rho)(\sigma) \to (\varphi \mapsto \rho)(\tau) \\
(\varphi \mapsto \rho)(\sigma_1\cap\cdots\cap\sigma_n) & = (\varphi \mapsto \rho)(\sigma_1) \cap \cdots \cap (\varphi \mapsto \rho)(\sigma_n) \\
(\varphi \mapsto \rho)(\forall\alpha.\sigma) & = \forall\alpha.(\varphi \mapsto \rho)(\sigma)
\end{array}$$

We will use $S$ to denote a generic substitution. Substitutions extend to bases in the natural way: $S(B) = \{x{:}S(\rho) \mid x{:}\rho \in B\}$.

For substitutions, the following properties hold:

*Property 2.6 Let S be a substitution.*

*i) If $\sigma \leq \tau$, then $S(\sigma) \leq S(\tau)$.*

*ii) If $B \leq B'$, then $S(B) \leq S(B')$.*

*Proof:* The second part is a consequence of the first, which is shown by induction on the definition of '$\leq$'. ■

**Expansion.** As mentioned above, the operation of expansion deals with the replacement of a sub-type of a type by an intersection of a number of renamed copies of that sub-type. When a sub-type is expanded, new type variables are generated, and other sub-types might be affected (e.g. the expansion of $\tau$ in $\sigma\to\tau$ might affect also $\sigma$: intuitively, each renamed copy of $\tau$ will have an associated copy of $\sigma$; see [36] for a detailed explanation). Ground types are not affected by expansions since all renamed copies coincide (and $\sigma\cap\sigma \sim \sigma$).

Two different definitions of expansion appear in the literature for LC, depending on whether one uses a set of types (see e.g. [36]) or a set of type variables (see e.g. [5]) to compute the set of types affected by the expansion. Our definition is inspired by [36]; the extension to deal with types containing sorts has already been done in [20], here quantifiers are also taken into account.

We consider expansions determined by three parameters: the sub-type to be expanded, the number of copies that have to be generated, and the set of types affected by the expansion. The third parameter is not present in standard definitions of expansion since it can be "computed", however we prefer to add it since it simplifies the definition. Of course, we will only apply an expansion to a type (or to a type derivation) if they are *compatible*, that is, if the third parameter of the expansion contains the corresponding set of affected types.

The types modified by the expansion $\langle\mu,n,A\rangle$ will be those that end with a type in $A$. The notion of *last sub-types* in a strict type plays an important role in this operation.

**Definition 2.7** The set of *last sub-types* of a type $\tau \in \mathcal{T}_s$, denoted by $last(\tau)$, is defined by:

$$\begin{array}{lcl} last(\varphi) & = & \{\varphi\} \\ last(s) & = & \{s\} \\ last(\sigma \to \rho) & = & \{\sigma \to \rho\} \cup last(\rho) \\ last(\forall \alpha.\sigma) & = & \{\forall \alpha.\sigma\} \cup last(\sigma[\varphi_\alpha/\alpha]). \end{array}$$

Note that for types of the form $\forall \alpha.\sigma$, according to our convention $\sigma$ is not a well-formed sub-type (free variables must belong to $\Phi$). For this reason we consider a mapping $\mathcal{A} \to \Phi$ that associates to each $\alpha$ a different fresh $\varphi_\alpha \in \Phi$, and rename $\alpha$ in $\sigma$, using $\varphi_\alpha$. In this way we can define sub-types of types in $\mathcal{T}$, as usual.

We define now the notion of compatibility between an expansion and a type derivation (it applies to types as particular case).

**Definition 2.8** (Compatibility) We represent a derivation $B \vdash_{\mathcal{E}} t : \sigma$ by a triple $\langle B, \sigma, E \rangle$ where $B$ is a basis, $\sigma$ a type, and $E$ the set of types assigned to the function symbols of $t$ in the derivation. An expansion $Ex_{\langle \mu, n, A \rangle}$ (where $\mu$ is a type in $\mathcal{T}$, $n \geq 2$, and $A$ a finite set of types) is *compatible with a derivation* represented by $\langle B, \sigma, E \rangle$ if the set $A$ contains the set $\mathcal{L}_\mu(\langle B, \sigma, E \rangle)$ defined as follows:

*i*) Any non-closed strict sub-type of $\mu$ is in $\mathcal{L}_\mu(\langle B, \sigma, E \rangle)$.

*ii*) Let $\tau$ be a non-closed strict (sub)type occurring in $\langle B, \sigma, E \rangle$. If $\tau'$ is a most general instance (with respect to the universal quantifiers) of $\tau$ such that $last(\tau') \cap \mathcal{L}_\mu(\langle B, \sigma, E \rangle) \neq \emptyset$, then $\tau' \in \mathcal{L}_\mu(\langle B, \sigma, E \rangle)$.

*iii*) Any non-closed strict sub-type of $\tau \in \mathcal{L}_\mu(\langle B, \sigma, E \rangle)$ is in $\mathcal{L}_\mu(\langle B, \sigma, E \rangle)$.

An expansion $Ex_{\langle \mu, n, A \rangle}$ is *compatible with a type* $\tau$ if it is compatible with the triple $\langle \emptyset, \tau, \emptyset \rangle$.

The definition of expansion, already non-trivial in the intersection system, becomes quite involved in the presence of universal quantifiers. We define it in two steps. First, given the set $A$ of types affected by the expansion, we see which are the variables that will need to be renamed (for free variables we use substitutions in order to do the renamings, but we also need to rename bound variables; those renamings are not substitutions according to our definition, but by abuse of terminology we call them substitutions as well). Then, if the expansion $\langle \mu, n, A \rangle$ is *compatible* with the type $\tau$ that we want to expand, we traverse $\tau$ top-down searching for maximal sub-types whose last sub-types are in $A$ or have an instance (obtained by replacing bound variables by types) in $A$. Those sub-types of $\tau$ will be replaced by intersections of renamed copies.

**Definition 2.9** (Expansion) Let $\mu$ be a type in $\mathcal{T}$, $n \geq 2$, and $A$ a finite set of types. The expansion determined by $\langle \mu, n, A \rangle$, denoted by $Ex_{\langle \mu, n, A \rangle}$, is defined as follows:

(*Renamings*): Let $\mathcal{V} = \{\varphi_1, \ldots, \varphi_m\}$ be the set of free type variables occurring in $A$, and let $S_i$ $(1 \leq i \leq n)$ be the substitution that replaces every $\varphi_j$ by a fresh variable $\varphi_j^i$, and every $\alpha_j$ and $\varphi_{\alpha_j}$ by $\alpha_j^i$ (actually, $S_i$ is just a renaming).

(*Expansion of a type*): For any $\tau \in \mathcal{T}$ (without loss of generality we assume that its bound variables are disjoint with those of $\mu, A$) compatible with the expansion $Ex_{\langle \mu, n, A \rangle}$, the type $Ex_{\langle \mu, n, A \rangle}(\tau)$ is obtained out of $\tau$ by traversing $\tau$ top-down and replacing in $\tau$ a maximal non-closed sub-type $\beta$ such that there exists a most general instance (w.r.t. the universal quantifiers) $\beta'$ of $\beta$ with $last(\beta') \cap A \neq \emptyset$

*a*) by $S_1(\beta) \cap \cdots \cap S_n(\beta)$ if $\beta' = \beta$,

*b*) otherwise by

$$\bigcap_{1 \leq j \leq p} (S_1(\beta'_j) \cap \cdots \cap S_n(\beta'_j) \cap \forall \overline{S_i(\overline{\alpha})}. Ex_{\langle \mu, n, A \rangle}(\rho[\overline{c_j}/\overline{\alpha}])[\overline{\alpha}/\overline{c_j}])$$

if $\beta = \forall\overline{\alpha}.\rho$, $\beta'_j$ $(1 \leq j \leq p)$ are all the most general instances of $\beta$ satisfying the condition, and $\overline{c_j}$ are fresh constants replacing the variables instantiated in the instance $\beta'_j$ of $\beta$.

(*Expansion of a derivation*): Let $B \vdash_{\mathcal{E}} t{:}\sigma$ be a type-derivation (represented by the triple $\langle B,\sigma,E\rangle$) compatible with $Ex_{\langle\mu,n,A\rangle}$. The result of the application of the expansion is then a triple:

$$\langle\{x{:}Ex_{\langle\mu,n,A\rangle}(\rho) \mid x{:}\rho \in B\}, Ex_{\langle\mu,n,A\rangle}(\sigma), \{Ex_{\langle\mu,n,A\rangle}(\rho) \mid \rho \in E\}\rangle$$

We will prove below that this triple represents a correct derivation (i.e. expansions are sound on derivations).

Some explanations are in order. The result of an operation of expansion is not unique because it depends on the choice of new variables in part (*Renamings*) of the definition; but it is unique modulo renaming of variables (and this is sufficient for our purpose). It is always a type in $\mathcal{T}$: we never introduce an intersection at the right-hand side of an arrow type, and never quantify an intersection type (see part (*Expansion of a type*)). A type might be affected by an expansion even if its free variables are disjoint with those of the sub-type to be expanded. The reason is that universally quantified variables represent an infinite set of terms (their instances), so if one instance is affected, the whole type is affected. If we are applying an expansion operation to a universally quantified type, some instances may be expanded (if their last sub-types are in the set of affected types) whereas others are not (if their last sub-types are not in this set). In this case the expansion of the universally quantified type will be the intersection of the expansions of each class of instances. Since there is only a finite set of affected types, the operation is well defined.

*Example 2.10* Let $\gamma$ be $(\varphi_1\to\varphi_2)\to(\varphi_3\to\varphi_1)\to\varphi_3\to\varphi_2$, and $Ex$ be the expansion determined by $\langle\varphi_1,2,\{\varphi_1,\varphi_3\to\varphi_1,\varphi_3\}\rangle$. First, we check that this expansion is compatible with $\gamma$: indeed, the set $\mathcal{L}_{\varphi_1}(\langle\emptyset,\gamma,\emptyset\rangle) = \{\varphi_1,\varphi_3\to\varphi_1,\varphi_3\}$. Then we compute the set of variables that will be renamed: $\mathcal{V} = \{\varphi_1,\varphi_3\}$. The result of the expansion of $\gamma$ is:

$$Ex(\gamma) = ((\varphi_1^1\cap\varphi_1^2)\to\varphi_2)\to((\varphi_3^1\to\varphi_1^1)\cap(\varphi_3^2\to\varphi_1^2))\to(\varphi_3^1\cap\varphi_3^2)\to\varphi_2.$$

Consider now a type with universal quantifiers and free variables, such as

$$\gamma = \forall\alpha_2\forall\alpha_3.(\varphi_1\to\alpha_2)\to(\alpha_3\to\varphi_1)\to\alpha_3\to\alpha_2,$$

Then $\mathcal{L}_{\varphi_1}(\langle\emptyset,\gamma,\emptyset\rangle) = \{\varphi_1,\forall\alpha_3.(\varphi_1\to\varphi_1)\to(\alpha_3\to\varphi_1)\to\alpha_3\to\varphi_1,$
$(\varphi_1\to\varphi_1)\to(\varphi_{\alpha_3}\to\varphi_1)\to\varphi_{\alpha_3}\to\varphi_1,\varphi_1\to\varphi_1,\varphi_{\alpha_3}\to\varphi_1,\varphi_{\alpha_3}\}$

Let $Ex$ be the expansion determined by $\langle\varphi_1,2,\mathcal{L}_{\varphi_1}(\langle\emptyset,\gamma,\emptyset\rangle)\rangle$, which is obviously compatible with $\gamma$. Then $\mathcal{V} = \{\varphi_1\}$ and

$$\begin{aligned}Ex(\gamma) = {} & (\forall\alpha_3.(\varphi_1^1\to\varphi_1^1)\to(\alpha_3\to\varphi_1^1)\to\alpha_3\to\varphi_1^1)\cap(\forall\alpha_3.(\varphi_1^2\to\varphi_1^2)\to(\alpha_3\to\varphi_1^2)\to\alpha_3\to\varphi_1^2)\cap\\ & (\forall\alpha_2\forall\alpha_3^1\forall\alpha_3^2.(\varphi_1^1\cap\varphi_1^2\to\alpha_2)\to((\alpha_3^1\to\varphi_1^1)\cap(\alpha_3^2\to\varphi_1^2))\to\alpha_3^1\cap\alpha_3^2\to\alpha_2).\end{aligned}$$

For an example with sorts, consider $\gamma = (\varphi_1\to s)\to\varphi_2$, and let $Ex$ be the expansion determined by $\langle\varphi_1\to s,2,\{\varphi_1\to s,\varphi_1\}\rangle$, which is compatible with $\gamma$ since $\mathcal{L}_{\varphi_1\to s}(\langle\emptyset,\gamma,\emptyset\rangle) = \{\varphi_1\to s,\varphi_1\}$. Then $\mathcal{V} = \{\varphi_1\}$ and $Ex(\gamma) = ((\varphi_1^1\to s)\cap(\varphi_1^2\to s))\to\varphi_2$.

If we apply instead the expansion determined by $\langle\varphi_2,2,\mathcal{L}_{\varphi_2}(\langle\emptyset,\gamma,\emptyset\rangle)\rangle$ to $\gamma$, where

$$\begin{aligned}\mathcal{L}_{\varphi_2}(\langle\emptyset,\gamma,\emptyset\rangle) &= \{\varphi_2,(\varphi_1\to s)\to\varphi_2,\varphi_1\to s,\varphi_1\}, \text{ and}\\ \mathcal{V} &= \{\varphi_2,\varphi_1\},\end{aligned}$$

we obtain $Ex(\gamma) = ((\varphi_1^1\to s)\to\varphi_2^1)\cap((\varphi_1^2\to s)\to\varphi_2^2)$.

For an operation of expansion the following property holds:

*Property 2.11* *Let $Ex = \langle \mu, n, A \rangle$ be an expansion compatible with a derivation represented by $\langle B,\sigma,E \rangle$. If $x{:}\rho \in B$ and $\rho \leq \sigma$, then $Ex(\rho) \leq Ex(\sigma)$.*

*Proof:* By induction on the definition of $\leq$. ■

**Lifting.** The operation of *lifting* replaces basis and type by a smaller basis and a larger type, in the sense of '$\leq$' (see [4] for details). This operation allows us to eliminate intersections and universal quantifiers, using the '$\leq$' relation. When applying a lifting to a derivation $B \vdash_{\mathcal{E}} t{:}\sigma$ we will use the pair $\langle B,\sigma \rangle$ to represent the derivation.

**Definition 2.12** (Lifting) An operation of *lifting* is determined by a pair $L = <\langle B_0, \tau_0 \rangle, \langle B_1, \tau_1 \rangle>$ such that $\tau_0 \leq \tau_1$ and $B_1 \leq B_0$, and $L(\langle B,\sigma \rangle) = \langle B',\sigma' \rangle$ where

$$\begin{array}{llll} \sigma' = \tau_1, \text{ if } \sigma = \tau_0, & & B' = B_1, \text{ if } B = B_0 \\ \sigma' = \sigma, \text{ otherwise} & & B' = B, \text{ otherwise} \end{array}$$

A lifting on types is determined by a pair $L = \langle \tau_0, \tau_1 \rangle$ such that $\tau_0 \leq \tau_1$ and is defined by

$$\begin{array}{rl} L(\sigma) & = \tau_1, \text{ if } \sigma = \tau_0 \\ & = \sigma, \text{ otherwise} \end{array}$$

**Closure.** The operation of *closure* introduces quantifiers, taking into account the basis where a type is used.

**Definition 2.13** (Closure) A *closure* is an operation characterised by a type-variable $\varphi$. It is defined by:

$$\begin{array}{rll} Cl_{\varphi}(\langle B,\sigma \rangle) & = \langle B, \forall \alpha.\sigma[\alpha/\varphi] \rangle, & \text{if } \varphi \text{ does not appear in } B \ (\alpha \text{ is a fresh variable}) \\ & = \langle B,\sigma \rangle, & \text{otherwise} \end{array}$$

It is extended to types by: $Cl_{\varphi}(\sigma) = (\tau)$, if $Cl_{\varphi}(\langle \emptyset,\sigma \rangle) = \langle \emptyset,\tau \rangle$.

**Chains of operations.** The *set Ch of chains* for types is defined as the smallest set containing expansions, substitutions, liftings, and closures on types, that is closed under composition.

**Definition 2.14** (Chains on types) *i*) A *chain* is an object $[O_1,\ldots,O_n]$, where each $O_i$ is an operation of substitution, expansion, lifting, or closure, and

$$[O_1,\ldots,O_n](\sigma) = O_1(\cdots(O_n(\sigma))\cdots).$$

*ii*) On chains the operation of concatenation is denoted by $*$, and:

$$[O_1,\ldots,O_i] * [O_{i+1},\ldots,O_n] = [O_1,\ldots,O_n].$$

*iii*) We say that $Ch_1 = Ch_2$, if for all $\sigma$, $Ch_1(\sigma) = Ch_2(\sigma)$.

*iv*) We extend the notion of compatibility to chains by: $Ch$ is compatible with $B \vdash_{\mathcal{E}} t{:}\sigma$ if, for all expansions $Ex$ that occur in $Ch$ (so $Ch = Ch_1 * [Ex] * Ch_2$), $Ex$ is compatible with $Ch_1(B \vdash_{\mathcal{E}} t{:}\sigma)$.

Notice that, although the operation of substitution seems redundant, in that one could simulate substitution via closure and lifting, this is only the case for type variables that *do not occur in the basis*; to instantiate type-variables that occur in the basis as well, substitution on types is essential.

### 2.2.2 Type Assignment Rules

We specify how to type terms through the presentation of type assignment rules. To deal with function symbols we use an environment $\mathcal{E}$ and the operations on types defined above: the types assigned to occurrences of function symbols are obtained from the type provided by the environment by making a chain of operations.

**Definition 2.15** (Type Assignment Rules) *i*) *Type assignment* (with respect to $\mathcal{E}$) is defined by the following natural deduction system in sequent form (where all types displayed are in $\mathcal{T}_s$, except for $\sigma_1,\ldots,\sigma_n$ in rule $(\mathcal{F})$ and $\sigma$ in rules $(\to E)$, $(\to I)$, and $(\leq)$). Note the use of a chain of operations in rule $(\mathcal{F})$.

$$(\leq):\frac{}{B\vdash_{\mathcal{E}} x{:}\tau}\,(a)\qquad (\to I):\frac{B,x{:}\sigma\vdash_{\mathcal{E}} t{:}\tau}{B\vdash_{\mathcal{E}}\lambda x.t{:}\sigma\to\tau}\qquad (\to E):\frac{B\vdash_{\mathcal{E}} t_1{:}\sigma\to\tau\quad B\vdash_{\mathcal{E}} t_2{:}\sigma}{B\vdash_{\mathcal{E}} Ap(t_1,t_2){:}\tau}$$

$$(\cap I):\frac{B\vdash_{\mathcal{E}} t{:}\sigma_1\quad\cdots\quad B\vdash_{\mathcal{E}} t{:}\sigma_n}{B\vdash_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n}\,(n\geq 1)\qquad (\mathcal{F}):\frac{B\vdash_{\mathcal{E}} t{:}\sigma_1\quad\cdots\quad B\vdash_{\mathcal{E}} t{:}\sigma_n}{B\vdash_{\mathcal{E}} F(t_1,\ldots,t_n){:}\sigma}\,(b)$$

$$(\forall I):\frac{B\vdash_{\mathcal{E}} t{:}\sigma}{B\vdash_{\mathcal{E}} t{:}\forall\alpha.\sigma[\alpha/\varphi]}\,(c)\qquad (\forall E):\frac{B\vdash_{\mathcal{E}} t{:}\forall\alpha.\sigma}{B\vdash_{\mathcal{E}} t{:}\sigma[\tau/\alpha]}$$

*a*) $x{:}\sigma\in B,\sigma\leq\tau$.

*b*) If there exists a chain $Ch$ on types such that $\sigma_1\to\cdots\to\sigma_n\to\sigma = Ch(\mathcal{E}(F))$.

*c*) If $\varphi$ does not occur (i.e. is not free) in $B$.

*ii*) If $B\vdash_{\mathcal{E}} t{:}\sigma$ is derivable in this system, we write $B\vdash_{\mathcal{E}} t{:}\sigma$.

As said before, the use of an environment in rule $(\mathcal{F})$ introduces a notion of polymorphism for our function symbols, which is an extension (with intersection types and general quantification) of the ML-style of polymorphism. The environment returns the 'principal type' for a function symbol; this symbol can be used with types that are 'instances' of its principal type, obtained by applying chains of operations.

Note that the rule $(\leq)$ is only defined for variables, and we have a $(\forall E)$-rule for arbitrary terms but not an $(\cap E)$-rule. Indeed, the $(\cap E)$-rule for arbitrary terms can be admitted to this system of rules without extending its expressive power. On the other hand, the $(\forall E)$-rule cannot be derived if it is not present in the system. This asymmetry comes from the fact that our types are strict with respect to '$\cap$', but not with respect to '$\forall$'.

*Example 2.16* We can derive $\vdash_{\mathcal{E}}\lambda xyz.x(yz){:}(\varphi_1\to\varphi_2)\to(\varphi_3\to\varphi_1)\to\varphi_3\to\varphi_2$ as follows, where $B=\{x{:}\varphi_1\to\varphi_2,y{:}\varphi_3\to\varphi_1,z{:}\varphi_3\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B\vdash_{\mathcal{E}} x{:}\varphi_1\to\varphi_2}\quad\dfrac{\dfrac{}{B\vdash_{\mathcal{E}} y{:}\varphi_3\to\varphi_1}\quad\dfrac{}{B\vdash_{\mathcal{E}} z{:}\varphi_3}}{B\vdash_{\mathcal{E}} yz{:}\varphi_1}}{B\vdash_{\mathcal{E}} x(yz){:}\varphi_2}}{B\backslash z\vdash_{\mathcal{E}}\lambda z.x(yz){:}\varphi_3\to\varphi_2}}{B\backslash y,z\vdash_{\mathcal{E}}\lambda yz.x(yz){:}(\varphi_3\to\varphi_1)\to\varphi_3\to\varphi_2}}{\vdash_{\mathcal{E}}\lambda xyz.x(yz){:}(\varphi_1\to\varphi_2)\to(\varphi_3\to\varphi_1)\to\varphi_3\to\varphi_2}$$

and also $\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha_3\alpha_2.(\varphi_1\to\alpha_2)\to(\alpha_3\to\varphi_1)\to\alpha_3\to\alpha_2$ with a derivation $D$ of the form:

$$\dfrac{\dfrac{\begin{array}{c}\boxed{\qquad\qquad\qquad\qquad\qquad\qquad}\\ \vdash_{\mathcal{E}} \lambda xyz.x(yz):(\varphi_1\to\varphi_2)\to(\varphi_3\to\varphi_1)\to\varphi_3\to\varphi_2\end{array}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha_2.(\varphi_1\to\alpha_2)\to(\varphi_3\to\varphi_1)\to\varphi_3\to\alpha_2}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha_3\alpha_2.(\varphi_1\to\alpha_2)\to(\alpha_3\to\varphi_1)\to\alpha_3\to\alpha_2}$$

Notice that the sub-derivation for $\vdash_{\mathcal{E}} \lambda xyz.x(yz):(\varphi_1\to\varphi_2)\to(\varphi_3\to\varphi_1)\to\varphi_3\to\varphi_2$ in $D$ is *exactly* the one given above, in the sense that no $\alpha$ appears there.

Now, performing the expansions of Example 2.10, we obtain the statements

$$\vdash_{\mathcal{E}} \lambda xyz.x(yz):((\varphi_1^1\cap\varphi_1^2)\to\varphi_2)\to((\varphi_3^1\to\varphi_1^1)\cap(\varphi_3^2\to\varphi_1^2))\to(\varphi_3^1\cap\varphi_3^2)\to\varphi_2$$

and

$$\vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_1\cap\sigma_2\cap\sigma_3$$

where $\sigma_1 = \forall\alpha_3.(\varphi_1^1\to\varphi_1^1)\to(\alpha_3\to\varphi_1^1)\to\alpha_3\to\varphi_1^1$,
$\sigma_2 = \forall\alpha_3.(\varphi_1^2\to\varphi_1^2)\to(\alpha_3\to\varphi_1^2)\to\alpha_3\to\varphi_1^2$, and
$\sigma_3 = \forall\alpha_2\forall\alpha_3^1\forall\alpha_3^2.(\varphi_1^1\cap\varphi_1^2\to\alpha_2)\to((\alpha_3^1\to\varphi_1^1)\cap(\alpha_3^2\to\varphi_1^2))\to\alpha_3^1\cap\alpha_3^2\to\alpha_2$,
which are derived as below (where, in the derivations, we will omit the premisse for rule $(\leq)$ (of the shape '$x{:}\sigma\in B$') as well as $\varphi$ for lack of space).

Take $B=\{x:\varphi_1^1\to\varphi_1^1, y:\varphi_3\to\varphi_1, z{:}\varphi_3\}$, then, for $\vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B\vdash_{\mathcal{E}} x:{}_1^1\to{}_1^1}\qquad\dfrac{\dfrac{}{B\vdash_{\mathcal{E}} y:{}_3\to{}_1^1}\qquad\dfrac{}{B\vdash_{\mathcal{E}} z:{}_3}}{B\vdash_{\mathcal{E}} yz:{}_1^1}}{B\vdash_{\mathcal{E}} x(yz):{}_1^1}}{B\backslash z\vdash_{\mathcal{E}} \lambda z.x(yz):{}_3\to{}_1^1}}{B\backslash y{,}z\vdash_{\mathcal{E}} \lambda yz.x(yz):({}_3\to{}_1)\to{}_3\to{}_1^1}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):({}_1^1\to{}_1^1)\to({}_3\to{}_1^1)\to{}_3\to{}_1^1}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha_3.({}_1^1\to{}_1^1)\to(\alpha_3\to{}_1^1)\to\alpha_3\to{}_1^1}$$

The derivation for $\vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_2$ is similar to the one above, just replace $\varphi_1^1$ by $\varphi_1^2$.

Take $B=\{x:(\varphi^1_1\cap\varphi^2_1)\rightarrow\varphi_2, y:(\varphi^1_3\rightarrow\varphi^1_1)\cap(\varphi^2_3\rightarrow\varphi^2_1), z:\varphi^1_3\cap\varphi^2_3\}$, then, for $\vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_3$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B\vdash_{\mathcal{E}} x:({}^1_1\cap{}^2_1)\rightarrow{}_2} \quad \dfrac{\dfrac{\dfrac{}{B\vdash_{\mathcal{E}} y:{}^1_3\rightarrow{}^1_1} \quad \dfrac{}{B\vdash_{\mathcal{E}} z:{}^1_3}}{B\vdash_{\mathcal{E}} yz:{}^1_1} \quad \dfrac{\dfrac{}{B\vdash_{\mathcal{E}} y:{}^2_3\rightarrow{}^2_1} \quad \dfrac{}{B\vdash_{\mathcal{E}} z:{}^2_3}}{B\vdash_{\mathcal{E}} yz:{}^2_1}}{B\vdash_{\mathcal{E}} yz:{}^1_1\cap{}^2_1}}{B\vdash_{\mathcal{E}} x(yz):{}_2}}{B\backslash z\vdash_{\mathcal{E}} \lambda z.x(yz):({}^1_3\cap{}^2_3)\rightarrow{}_2}}{B\backslash y,z\vdash_{\mathcal{E}} \lambda yz.x(yz):(({}^1_3\rightarrow{}^1_1)\cap({}^2_3\rightarrow{}^2_1))\rightarrow({}^1_3\cap{}^2_3)\rightarrow{}_2}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):(({}^1_1\cap{}^2_1)\rightarrow{}_2)\rightarrow(({}^1_3\rightarrow{}^1_1)\cap({}^2_3\rightarrow{}^2_1))\rightarrow({}^1_3\cap{}^2_3)\rightarrow{}_2}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha^2_3.({}^1_1\cap{}^2_1\rightarrow{}_2)\rightarrow(({}^1_3\rightarrow{}^1_1)\cap(\alpha^2_3\rightarrow{}^2_1))\rightarrow{}^1_3\cap\alpha^2_3\rightarrow{}_2}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha^1_3\forall\alpha^2_3.({}^1_1\cap{}^2_1\rightarrow{}_2)\rightarrow((\alpha^1_3\rightarrow{}^1_1)\cap(\alpha^2_3\rightarrow{}^2_1))\rightarrow\alpha^1_3\cap\alpha^2_3\rightarrow{}_2}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha_2\forall\alpha^1_3\forall\alpha_3.({}^1_1\cap{}^2_1\rightarrow\alpha_2)\rightarrow((\alpha^1_3\rightarrow{}^1_1)\cap(\alpha^2_3\rightarrow{}^2_1))\rightarrow\alpha^1_3\cap\alpha^2_3\rightarrow\alpha_2}$$

Finally, combining these three derivations, we obtain:

$$\dfrac{\begin{array}{c}\boxed{\qquad\qquad\qquad}\\ \vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_1\end{array} \quad \begin{array}{c}\boxed{\qquad\qquad\qquad}\\ \vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_2\end{array} \quad \begin{array}{c}\boxed{\qquad\qquad\qquad}\\ \vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_3\end{array}}{\vdash_{\mathcal{E}} \lambda xyz.x(yz):\sigma_1\cap\sigma_2\cap\sigma_3}\;(\cap I)$$

Similarly, we can build a derivation for

$$\vdash_{\mathcal{E}} \lambda xyz.x(yz):\forall\alpha_3\alpha_2\alpha_1.(\alpha_1\rightarrow\alpha_2)\rightarrow(\alpha_3\rightarrow\alpha_1)\rightarrow\alpha_3\rightarrow\alpha_2$$

which is in fact the principal type for this term. We can also define a function $F$ which plays the same role as this $\lambda$-term, using a rewrite rule

$$F(x,y,z)\rightarrow Ap(x,Ap(y,z)).$$

The term $F(x,y,z)$ is typeable with respect to an environment where

$$\begin{array}{rcl}\mathcal{E}(F) &=& \forall\alpha_3\alpha_2\alpha_1.(\alpha_1\rightarrow\alpha_2)\rightarrow(\alpha_3\rightarrow\alpha_1)\rightarrow\alpha_3\rightarrow\alpha_2 \text{ or}\\ \mathcal{E}(F) &=& (\varphi_1\rightarrow\varphi_2)\rightarrow(\varphi_3\rightarrow\varphi_1)\rightarrow\varphi_3\rightarrow\varphi_2.\end{array}$$

## 2.3 Soundness of operations on derivations

The four operations defined above are sound in the sense that, when applied to a derivation, they yield a derivation. We will show this result for each of the individual operations.

**Theorem 2.17** (Soundness of Substitution) *Let $S$ be a substitution and $\mathcal{E}$ an environment, then $B\vdash_{\mathcal{E}} t:\sigma$ implies $S(B)\vdash_{\mathcal{E}} t:S(\sigma)$.*

*Proof:* By induction on the structure of derivations. The only interesting cases are:

$(\leq)$: Then $t\equiv x$ and $B\leq\{x:\sigma\}$. By Lemma 2.6(*ii*), $S(B)\leq\{x:S(\sigma)\}$, so $S(B)\vdash_{\mathcal{E}} x:S(\sigma)$.

$(\mathcal{F})$: Then $t\equiv F(t_1,\ldots,t_n)$, there are $\sigma_1,\ldots,\sigma_n\in\mathcal{T}$ and a chain $Ch$ such that, for every $1\leq i\leq n$, $B\vdash_{\mathcal{E}} t_i:\sigma_i$ and $Ch(\mathcal{E}(F))=\sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\sigma$. Then by induction, $S(B)\vdash_{\mathcal{E}} t_i:S(\sigma_i)$, for every $1\leq i\leq n$; since $[S]*Ch$ is a chain and

$$([S]*Ch)(\mathcal{E}(F))=S(\sigma_1)\rightarrow\cdots\rightarrow S(\sigma_n)\rightarrow S(\sigma),$$

by $(\mathcal{F})$ also $S(B) \vdash_{\mathcal{E}} t{:}S(\sigma)$.

$(\forall I)$: Then there exists a $\tau$ such that $\sigma = \forall\alpha.\tau[\alpha/\varphi]$, $B \vdash_{\mathcal{E}} t{:}\tau$, and $\varphi$ does not occur (free) in $B$. By induction, $S(B) \vdash_{\mathcal{E}} t{:}S(\tau)$. Without loss of generality, we can assume that $S(\varphi) = \varphi$, then $\forall\alpha.S(\tau)[\alpha/\varphi] = S(\forall\alpha.\tau[\alpha/\varphi])$. Therefore, if $\varphi$ occurs in $\tau$, it also occurs in $S(\tau)$, and it will not occur in $S(B)$. But then, by rule $(\forall I)$, also $S(B) \vdash_{\mathcal{E}} t{:}\forall\alpha.S(\tau)[\alpha/\varphi]$.

$(\forall E)$: Then there are $\tau,\rho$ such that $\sigma = \tau[\rho/\alpha]$, and $B \vdash_{\mathcal{E}} t{:}\forall\alpha.\tau$. By induction, $S(B) \vdash_{\mathcal{E}} t{:}S(\forall\alpha.\tau)$. Since $S$ does not affect bound variables, also $S(B) \vdash_{\mathcal{E}} t{:}\forall\alpha.S(\tau)$. Then, by rule $(\forall E)$, we get $S(B) \vdash_{\mathcal{E}} t{:}S(\tau)[S(\rho)/\alpha]$, and $S(\tau)[S(\rho)/\alpha] = S(\tau[\rho/\alpha])$.

**Theorem 2.18** (Soundness of Lifting) *Let $L$ be a lifting and $\mathcal{E}$ an environment, then $B \vdash_{\mathcal{E}} t{:}\sigma$ implies $L(B) \vdash_{\mathcal{E}} t{:}L(\sigma)$.*

*Proof:* Notice that $L(B) \leq B$ and $\sigma \leq L(\sigma)$. The proof is done by induction on the structure of derivations.

$(\leq)$: Then $t \equiv x$ and there exists a $\tau$ such that $x{:}\tau \in B$, with $\tau \leq \sigma$. Since $L(B) \leq B$, there exists a $\rho$ such that $x{:}\rho \in L(B)$, and $\rho \leq \tau \leq \sigma \leq L(\sigma)$. But then, by rule $(\leq)$, $L(B) \vdash_{\mathcal{E}} x{:}L(\sigma)$.

$(\mathcal{F})$: Then $t = F(t_1,\ldots,t_n)$, and there are $\sigma_1,\ldots,\sigma_n$ such that, for $1 \leq i \leq n$, $B \vdash_{\mathcal{E}} t_i{:}\sigma_i$, and there exists a chain $Ch$ such that $Ch(\mathcal{E}(F)) = \sigma_1 \to \cdots \to \sigma_n \to \sigma$. Since $L(B) \leq B$, by induction $L(B) \vdash_{\mathcal{E}} t_i{:}\sigma_i$, for $1 \leq i \leq n$. Since $\sigma \leq L(\sigma)$,

$$L = <\langle \emptyset, \sigma_1 \to \cdots \to \sigma_n \to \sigma \rangle, \langle \emptyset, \sigma_1 \to \cdots \to \sigma_n \to L(\sigma) \rangle>$$

is a lifting, so $([L] * Ch)\mathcal{E}(F) = \sigma_1 \to \cdots \to \sigma_n \to L(\sigma)$, and $L(B) \vdash_{\mathcal{E}} F(t_1,\ldots,t_n){:}L(\sigma)$ by rule $(\mathcal{F})$.

$(\to E)$: Then $t = Ap(t_1,t_2)$, and there is a $\rho$ such that $B \vdash_{\mathcal{E}} t_1{:}\rho \to \sigma$, and $B \vdash_{\mathcal{E}} t_2{:}\rho$. Since $\sigma \leq L(\sigma)$, also $\rho \to \sigma \leq \rho \to L(\sigma)$, so by induction, $L(B) \vdash_{\mathcal{E}} t_1{:}\rho \to L(\sigma)$ and, by rule $(\to E)$, also $L(B) \vdash_{\mathcal{E}} Ap(t_1,t_2){:}L(\sigma)$.

$(\to I)$: Then $t = \lambda x.t'$, and there are $\rho,\mu$ such that $\sigma = \rho \to \mu$ and $B, x{:}\rho \vdash_{\mathcal{E}} t'{:}\mu$, and $\delta,\gamma$ such that $L(\sigma) = \gamma \to \delta$, and $\gamma \leq \rho, \mu \leq \delta$. Then, by induction $L(B), x{:}\gamma \vdash_{\mathcal{E}} t'{:}\delta$, and therefore, by rule $(\to I)$, also $L(B) \vdash_{\mathcal{E}} \lambda x.t'{:}\gamma \to \delta$.

$(\forall I)$: Then $\sigma = \forall\alpha.\rho[\alpha/\varphi]$, and $B \vdash_{\mathcal{E}} t{:}\rho$. Since $\forall\alpha.\rho[\alpha/\varphi] \leq L(\sigma)$, by definition of '$\leq$', either:

$(L(\sigma) = \rho[\mu/\varphi])$: By induction, $L(B) \vdash_{\mathcal{E}} t{:}\rho$, and, using rule $(\forall I)$ we obtain $L(B) \vdash_{\mathcal{E}} t{:}\forall\alpha.\rho[\alpha/\varphi]$ (without loss of generality we assume that $\varphi$ does not occur in $L(B)$). Now, using $(\forall E)$ we obtain $L(B) \vdash_{\mathcal{E}} t{:}\rho[\mu/\varphi]$.

$(L(\sigma) = \forall\alpha'.\sigma$, *with $\alpha'$ fresh*$)$: By induction, $L(B) \vdash_{\mathcal{E}} t{:}\rho$, and, since $\alpha'$ is fresh, $L(B) \vdash_{\mathcal{E}} t{:}\forall\alpha'.\sigma$, by rule $(\forall I)$.

$(L(\sigma) = \forall\alpha.\mu[\alpha/\varphi]$, *with $\rho \leq \mu$*$)$: Then, by induction, $L(B) \vdash_{\mathcal{E}} t{:}\mu$, and $L(B) \vdash_{\mathcal{E}} t{:}\forall\alpha.\mu[\alpha/\varphi]$ by rule $(\forall I)$.

$(\cap I)$: Then $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, and, for $1 \leq i \leq n$, $B \vdash_{\mathcal{E}} t{:}\sigma_i$. Then there is an $1 \leq i \leq n$, such that $\sigma_i \leq L(\sigma)$. Then, by induction, $L(B) \vdash_{\mathcal{E}} t{:}L(\sigma)$. ■

A direct consequence of this theorem is that the following derivation rule is admissible.

$$\frac{B \vdash_{\mathcal{E}} t{:}\sigma}{B \vdash_{\mathcal{E}} t{:}\tau}\ (\sigma \leq \tau)$$

Also, the following is immediate.

*Corollary 2.19* $B \vdash_{\mathcal{E}} t{:}\sigma_1 \cap \cdots \cap \sigma_n$, *if and only if* $B \vdash_{\mathcal{E}} t{:}\sigma_i$, *for all* $1 \leq i \leq n$.

**Theorem 2.20** (Soundness of Expansion) *Let $Ex = \langle \mu, n, A \rangle$ be an expansion compatible with the derivation $B \vdash_{\mathcal{E}} t{:}\sigma$ represented by $\langle B,\sigma,E \rangle$. If $Ex(\langle B,\sigma,E \rangle) = \langle B',\sigma',E' \rangle$, then $B' \vdash_{\mathcal{E}} t{:}\sigma'$.*

*Proof:* By induction on the structure of derivations.

$(\leq)$: Then $t \equiv x$ and there exists $\tau$ such that $x{:}\tau \in B$, with $\tau \leq \sigma$. By Property 2.11, $Ex(\tau) \leq Ex(\sigma)$, then by rule $(\leq)$, $B' \vdash_{\mathcal{E}} x{:}\sigma'$.

$(\mathcal{F})$: Then $t = F(t_1,\ldots,t_m)$, and there are $\sigma_1,\ldots,\sigma_m$ such that, for $1 \leq i \leq m$, $B \vdash_{\mathcal{E}} t_i{:}\sigma_i$, and there exists a chain $Ch$ such that $Ch(\mathcal{E}(F)) = \sigma_1 \to \cdots \to \sigma_m \to \sigma$. We distinguish three cases:

*a*) If $\sigma' = Ex(\sigma) = \bigcap_{1 \leq j \leq n} S_j(\sigma)$ then $Ex(\sigma_i) = \bigcap_{1 \leq j \leq n} S_j(\sigma_i)$ and

$$Ex(\sigma_1 \to \cdots \to \sigma_m \to \sigma) = \bigcap_{1 \leq j \leq n} S_j(\sigma_1 \to \cdots \to \sigma_m \to \sigma),$$

since the expansion is compatible with the derivation. By induction, $Ex(B) \vdash_{\mathcal{E}} t_i{:}Ex(\sigma_i)$. Therefore, by Corollary 2.19, $Ex(B) \vdash_{\mathcal{E}} t_i{:}S_j(\sigma_i)$, and using rule $(\mathcal{F})$ with $[S_j] * Ch$, we can derive $Ex(B) \vdash_{\mathcal{E}} t{:}S_j(\sigma)$. We then obtain $B' \vdash_{\mathcal{E}} t{:}\sigma'$ using rule $(\cap I)$.

*b*) If $Ex(Ch(\mathcal{E}(F))) = Ex(\sigma_1) \to \ldots \to Ex(\sigma_n) \to Ex(\sigma)$ then, since by induction $Ex(B) \vdash_{\mathcal{E}} t_i{:}Ex(\sigma_i)$, we can apply rule $(\mathcal{F})$ with a chain $[Ex] * Ch$ to obtain $B' \vdash_{\mathcal{E}} t{:}\sigma'$.

*c*) Otherwise, $\sigma$ is a universally quantified type where some instances are expanded as in part *(a)* and some instances as in part *(b)*. Then $Ex(\sigma)$ is an intersection type where the derivation for each component can be obtained as in part *(a)* and part *(b)*, and combined using rule $(\cap I)$.

$(\to E)$: Then $t = Ap(t_1,t_2)$, and there is a $\rho$ such that $B \vdash_{\mathcal{E}} t_1{:}\rho \to \sigma$, and $B \vdash_{\mathcal{E}} t_2{:}\rho$. By induction, $Ex(B) \vdash_{\mathcal{E}} t_1{:}Ex(\rho \to \sigma)$, and $Ex(B) \vdash_{\mathcal{E}} t_2{:}Ex(\rho)$. We consider two cases:

*a*) If $Ex(\rho \to \sigma) = \bigcap_{1 \leq j \leq n} S_j(\rho \to \sigma)$ then also $Ex(\rho) = \bigcap_{1 \leq j \leq n} S_j(\rho)$ and $Ex(\sigma) = \bigcap_{1 \leq j \leq n} S_j(\sigma)$, since the expansion is compatible with the derivation. Then, by Corollary 2.19, $Ex(B) \vdash_{\mathcal{E}} t_1{:}S_j(\rho \to \sigma)$, and $Ex(B) \vdash_{\mathcal{E}} t_2{:}S_j(\rho)$. Therefore, using rules $(\to E)$ and $(\cap I)$ we obtain $Ex(B) \vdash_{\mathcal{E}} Ap(t_1,t_2){:}Ex(\sigma)$.

*b*) Otherwise, $Ex(\rho \to \sigma) = Ex(\rho) \to Ex(\sigma)$. Then $Ex(B) \vdash_{\mathcal{E}} Ap(t_1,t_2){:}Ex(\sigma)$ follows by induction using rule $(\to E)$.

$(\to I)$: Then $t = \lambda x.t'$, and there are $\rho,\mu$ such that $\sigma = \rho \to \mu$ and $B,x{:}\rho \vdash_{\mathcal{E}} t'{:}\mu$. Then, by induction $Ex(B),x{:}Ex(\rho) \vdash_{\mathcal{E}} t'{:}Ex(\mu)$. If $Ex(\rho \to \mu)$ is an intersection of renamed copies, we proceed as in the previous cases. Otherwise, $Ex(\rho \to \mu) = Ex(\rho) \to Ex(\mu)$. Since by induction $Ex(B),x{:}Ex(\rho) \vdash_{\mathcal{E}} t'{:}Ex(\mu)$, using rule $(\to I)$ we obtain $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\sigma)$.

$(\forall I)$: Then $\sigma = \forall \alpha.\rho[\alpha/\varphi]$, and $B \vdash_{\mathcal{E}} t{:}\rho$, where $\varphi$ does not occur in $B$. If $Ex(\sigma)$ is an intersection of renamed copies of $\sigma$, then so is $Ex(\rho)$, and by induction $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\rho)$. Using Corollary 2.19 and rule $(\forall I)$ we obtain $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\sigma)$. Otherwise, there is an instance of $\sigma$ (with respect to the quantifiers) whose expansion is not an intersection of renamed copies. Since, without loss of generality, we can assume that $\varphi$ is not in $A$, also the expansion of $\rho$ is in $\mathcal{T}_s$ in this case. Then, by induction and rule $(\forall I)$ we obtain $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\sigma)$.

$(\forall E)$: Then $\sigma = \sigma'[\tau/\alpha]$ and $B \vdash_{\mathcal{E}} t{:}\forall \alpha.\sigma'$.
By induction, $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\forall \alpha.\sigma')$. Since all the instances of $\forall \alpha.\sigma'$ are taken into account in $Ex(\forall \alpha.\sigma')$, in particular we obtain $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\sigma)$ using $(\forall E)$.

$(\cap I)$: Then $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, and, for $1 \leq i \leq n$, $B \vdash_{\mathcal{E}} t{:}\sigma_i$. Then, by induction and rule $(\cap I)$ we deduce $Ex(B) \vdash_{\mathcal{E}} t{:}Ex(\sigma)$. ∎

**Theorem 2.21** (Soundness of Closure) *Let $Cl = \langle \varphi \rangle$ be a closure such that $Cl(\langle B,\sigma \rangle) = \langle B',\sigma' \rangle$, and $\mathcal{E}$ an environment. Then $B \vdash_{\mathcal{E}} t{:}\sigma$ implies $B' \vdash_{\mathcal{E}} t{:}\sigma'$.*

*Proof:* Direct by definition of closure, using rule $(\forall I)$. ∎

We then have:

**Theorem 2.22** (Soundness of Chains) *Let $B \vdash_{\mathcal{E}} t{:}\sigma$, $Ch$ be a compatible chain, and $\mathcal{E}$ an environment. Then $Ch(B \vdash_{\mathcal{E}} t{:}\sigma) = Ch(B) \vdash_{\mathcal{E}} t{:}Ch(\sigma)$.*

### 2.3.1 Type Assignment for Rewrite Rules

Being able to infer a type for a term does not give any guarantee about the typing of the terms in any reduction path out of it. Indeed we need to make sure that the rewrite rules respect the intended functional behaviour for the function symbols of the signature expressed by the environment. The environment, however, does not express a strict condition on the type of function symbols, leaving room to flexibility by letting us use different consistent instances of the type of a symbol for different occurrences of it. So, we would like to have a certain degree of flexibility also in the use of rewriting rules, without losing the property of subject reduction, which is essential in type systems. In order to achieve this we define a notion of type assignment on rewrite rules, as done in [8], using the notion of principal pair (also called principal typing). The typeability of rules ensures consistency with respect to the environment.

**Definition 2.23** (Principal pair) $\langle P,\pi\rangle$ is called a *principal pair for $t$ with respect to $\mathcal{E}$*, if $P \vdash_{\mathcal{E}} t{:}\pi$, and for all $B \vdash_{\mathcal{E}} t{:}\sigma$, there is a chain $Ch$ compatible with $P \vdash_{\mathcal{E}} t{:}\pi$ such that $Ch(\langle P,\pi\rangle) = \langle B,\sigma\rangle$.

**Definition 2.24** *i*) We say that $l \to r \in \mathbf{R}$ with defined symbol $F$ *is typeable with respect to $\mathcal{E}$*, if there are $P$, and $\pi \in \mathcal{T}$ such that:

- *a*) $\langle P,\pi\rangle$ is a principal pair for $l$ with respect to $\mathcal{E}$, and each chain $Ch$ compatible with $P \vdash_{\mathcal{E}} l{:}\pi'$ is compatible with $P \vdash_{\mathcal{E}} r{:}\pi$.
- *b*) In $P \vdash_{\mathcal{E}} l{:}\pi$ and $P \vdash_{\mathcal{E}} r{:}\pi$ all occurrences of $F$ in side conditions to rule $(\mathcal{F})$ are typed with $\mathcal{E}(F)$.

*ii*) We say that $(\Sigma,\mathbf{R})$ *is typeable with respect to $\mathcal{E}$*, if all $\mathbf{r} \in \mathbf{R}$ are.

Notice that, by the formulation of part (*i.a*), the set of types permitted to occur in the derivation for $P \vdash_{\mathcal{E}} r{:}\pi$ is restricted.

Note that for a rule $F(t_1,\ldots,t_n) \to r$ to be typeable, $\mathcal{E}(F)$ must be of the form $\sigma_1{\to}\ldots{\to}\sigma_n{\to}\sigma$. Although $\mathcal{E}(F)$ cannot have an outermost universal quantifier, its free variables play the same role as universally quantified variables (since they can be instantiated by substitution operations). In particular, for the polymorphic identity function $I$ we will use $\mathcal{E}(I) = \varphi{\to}\varphi$.

*Example 2.25* We show now the type assignment for the rewrite rule $D(x) \to Ap(x,x)$ in an environment $\mathcal{E}$ where $\mathcal{E}(D) = (\varphi_1{\to}\varphi_2){\cap}\varphi_1{\to}\varphi_2$. Let $B = \{x{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1\}$, then

$$\dfrac{\dfrac{\dfrac{}{B \vdash_{\mathcal{E}} x{:}\varphi_1{\to}\varphi_2} \qquad \dfrac{}{B \vdash_{\mathcal{E}} x{:}\varphi_1}}{B \vdash_{\mathcal{E}} x{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1}}{B \vdash_{\mathcal{E}} D(x){:}\varphi_2}$$

Indeed, this is a principal derivation for $D(x)$. In order to type the rewrite rule we have to show that $\{x{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1\} \vdash_{\mathcal{E}} Ap(x,x){:}\varphi_2$, which is easy: let $B = \{x{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1\}$, then:

$$\dfrac{\dfrac{}{B \vdash_{\mathcal{E}} x{:}(\varphi_1{\to}\varphi_2)} \qquad \dfrac{}{B \vdash_{\mathcal{E}} x{:}\varphi_1}}{B \vdash_{\mathcal{E}} Ap(x,x){:}\varphi_2}$$

We will only consider TRS $+\beta$ that are typeable with respect to a given environment $\mathcal{E}$.

### 2.3.2 Subject Reduction

We will now show that reductions preserve types in our system. In the proof of Subject Reduction we will use one more lemma:

*Lemma 2.26* *Let $\mathcal{E}$ be an environment, $t$ a term, and* R *a term-substitution.*

*i) If $B \vdash_{\mathcal{E}} t{:}\sigma$ and $B'$ is a basis such that $B' \vdash_{\mathcal{E}} x^{\mathrm{R}}{:}\rho$ for every statement $x{:}\rho \in B$, then $B' \vdash_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma$.*

*ii) If there are $B$ and $\sigma$ such that $B \vdash_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma$, then for every $x$ occurring in $t$ there is a type $\rho_x$ such that $\{x{:}\rho_x \mid x \text{ occurs in } t\} \vdash_{\mathcal{E}} t{:}\sigma$, and $B \vdash_{\mathcal{E}} x^{\mathrm{R}}{:}\rho_x$.*

By induction on the structure of derivations. ■

**Theorem 2.27** (Subject Reduction Theorem) *If $B \vdash_{\mathcal{E}} t{:}\sigma$ and $t \rightarrow t'$, then $B \vdash_{\mathcal{E}} t'{:}\sigma$.*

For a $\beta$-reduction step the proof is standard, so we consider only the case of a rewrite step. Let $l \rightarrow r$ be the typeable rewrite rule applied in the rewrite step $t \rightarrow t'$. We will prove that for every term-substitution R and type $\mu$, if $B \vdash_{\mathcal{E}} l^{\mathrm{R}}{:}\mu$, then $B \vdash_{\mathcal{E}} r^{\mathrm{R}}{:}\mu$, which proves the theorem.
Since $\mathbf{r}$ is typeable, there are $P, \pi$ such that $\langle P, \pi \rangle$ is a principal pair for $l$ with respect to $\mathcal{E}$, and $P \vdash_{\mathcal{E}} r{:}\pi$. Suppose R is a term-substitution such that $B \vdash_{\mathcal{E}} l^{\mathrm{R}}{:}\mu$. By Lemma 2.26(*ii*) there is a $B'$ such that for every $x{:}\rho \in B'$, $B \vdash_{\mathcal{E}} x^{\mathrm{R}}{:}\rho$, and $B' \vdash_{\mathcal{E}} l{:}\mu$. Since $\langle P, \pi \rangle$ is a principal typing for $l$ with respect to $\mathcal{E}$, by Definition 2.23 there is a chain $Ch$ compatible with $\langle P, \pi \rangle$ such that $Ch(\langle P, \pi \rangle) = \langle B', \mu \rangle$. Since $P \vdash_{\mathcal{E}} r{:}\pi$, by Theorem 2.22 also $B' \vdash_{\mathcal{E}} r{:}\mu$. Then by Lemma 2.26(*i*) $B \vdash_{\mathcal{E}} r^{\mathrm{R}}{:}\mu$. ■

## 3 Strong Normalisation

As mentioned in the introduction, types serve not only as specifications and as a way to ensure that programs 'cannot go wrong' during execution, but also to ensure that computations terminate. In fact, this is a well-known property of the intersection system for LC, and of System F, but the situation is different in TRS (a rule $t \rightarrow_{\mathbf{r}} t$ may be typeable, although it is obviously non-terminating). In this section, we will focus on the restrictions necessary to obtain a strong normalisation result.

### 3.1 The General Scheme

Inspired by the work of Jouannaud and Okada [26], who defined a general scheme of recursion that ensures termination of higher-order rewrite rules combined with LC, we will define a general scheme for TRS $+\beta$, such that typeability of $(\Sigma, \mathbf{R})$ in the (second-order) polymorphic intersection system defined in this paper implies strong normalisation of all typeable terms.

**Definition 3.1** (General Scheme of Recursion) Let $\Sigma$ be a signature with a set of function symbols $\mathcal{F}_n = \mathcal{Q} \cup \{F^1, \ldots, F^n\}$, where $F^1, \ldots, F^n$ will be the defined symbols, and $\mathcal{Q}$ the set of constructors. We will assume that $F^1, \ldots, F^n$ are defined incrementally (i.e. there is no mutual recursion), by typeable rules that satisfy the *general scheme*:

$$F^i\,(\overline{\mathrm{C}[\overline{x}]}, \overline{y}) \rightarrow \mathrm{C}'[F^i\,(\overline{\mathrm{C}1\overline{x}}, \overline{y}), \ldots, F^i\,(\overline{\mathrm{C}_m[\overline{x}]}, \overline{y}), \overline{y}],$$

where $\overline{x}, \overline{y}$ are sequences of variables such that $\overline{x} \subseteq \overline{y}$; $\overline{\mathsf{C}[\;]}$, $\mathsf{C}'[\;]$, $\overline{\mathsf{C}1}, \ldots, \overline{\mathsf{C}_m[\;]}$ are sequences of contexts in $T(\mathcal{F}_{i-1}, \mathcal{X})$; and for every $1 \leq j \leq m$, $\overline{\mathsf{C}[\overline{x}]} \rhd_{mul} \overline{\mathsf{C}_j[\overline{x}]}$, where $\lhd$ is the strict sub-term ordering (i.e. $\rhd$ denotes strict super-term) and *mul* denotes multi-set extension. Moreover, in the principal derivation $P \vdash_{\mathcal{E}} F^i(\overline{\mathsf{C}[\overline{x}]}, \overline{y}) : \pi$ of $F^i(\overline{\mathsf{C}[\overline{x}]}, \overline{y})$, the types associated to the variables $\overline{y}$ in $P$ are the types of the corresponding arguments of $F^i$ in $\mathcal{E}(F^i)$.

This general scheme is a generalisation of primitive recursion. It imposes two main restrictions on the definition of functions: the terms in the multi-sets $\overline{\mathsf{C}_j[\overline{x}]}$ are sub-terms of terms in $\overline{\mathsf{C}[\overline{x}]}$ (this is the 'primitive recursive' aspect of the scheme), and the variables $\overline{x}$ must also appear as arguments in the left-hand side of the rule. Both restrictions are essential in the proof of the Strong Normalisation Theorem below. The last one can be replaced by a typing condition, requiring that the variables in $\overline{x}$ that are not included in $\overline{y}$ can only be assigned base types. Also, instead of the multi-set extension of the subterm ordering, a lexicographic extension can be used, or even a combination of lexicographic and multi-set (see [21] for details about these variants of the scheme).

Note that although the general scheme has a primitive recursive aspect, it allows the definition of non-primitive functions thanks to the higher-order features available in TRS $+ \beta$: for example, Ackermann's function can be represented.

$$\begin{array}{lll} h(0) & \to & \lambda x.\mathsf{Succ}(x) \\ h(\mathsf{Succ}(x)) & \to & \lambda y.H(h(x), y) \\ \\ H(g, 0) & \to & Ap(g, \mathsf{Succ}(0)) \\ H(g, \mathsf{Succ}(y)) & \to & Ap(g, H(g, y)) \end{array}$$

where $\mathsf{Succ}$ is the successor function.

Also the rewrite rules of Combinatory Logic are *not* recursive, so, in particular, satisfy the scheme.

## 3.2 The strong normalisation theorem

We shall prove that, when the rewrite rules satisfy the general schema, every typeable term is strongly normalisable. This will be done using Tait-Girard's method [22] and the techniques devised in [26] in order to cope with some of the difficulties that the presence of algebraic rewriting makes arise.

From now on all the rewrite rules will be assumed to satisfy the general schema.

In the following, a sequence $e_1, \ldots, e_n$ of elements will be denoted by $\overline{e}$. The length of the sequence will be denoted by $|\overline{e}|$ (so $|e_1, \ldots, e_n| = n$). In this section we shall not distinguish between free and bound type variables. Type variables will be denoted by $\varphi, \varphi', \varphi_1, \ldots$. Recall that a term is called *neutral* if it is not an abstraction.

**Definition 3.2** A *Reducibility Candidate* of type $\tau$ is a set $\mathcal{R}^\tau$ of terms typeable with $\tau$ and such that:

$(C1)$: If $t \in \mathcal{R}^\tau$, then $t$ is strongly normalisable, $\mathcal{SN}(t)$.

$(C2)$: If $t \in \mathcal{R}^\tau$ and $t \to^* t'$, then $t' \in \mathcal{R}^\tau$.

$(C3)$: If $t$ is neutral and typeable with type $\tau$, and if, for every $u$, $t \to u$ implies $u \in \mathcal{R}^\tau$, then $t \in \mathcal{R}^\tau$.

Note that any reducibility candidate contains all the term-variables and that, for any type $\rho$, $\mathcal{SN}^\rho$ is a reducibility candidate.

**Definition 3.3** Let $\rho$ be a type, and let $\overline{\mathcal{R}^{\gamma}}$ be a sequence of reducibility candidates $\mathcal{R}_1^{\gamma_1},\ldots,\mathcal{R}_n^{\gamma_n}$ such that $|\overline{\mathcal{R}^{\gamma}}| = |\overline{\varphi}|$, where $\{\overline{\varphi}\} \supseteq FV(\rho)$; then we can define the set of terms $\mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ by induction on $\rho$, as follows:

$(\rho \equiv s)$: $\mathsf{Red}^{s}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] = \mathcal{SN}^{s}$.

$(\rho \equiv \varphi_i)$: $\mathsf{Red}^{\varphi_i}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] = \mathcal{R}_i^{\gamma_i}$.

$(\rho \equiv \sigma \cap \tau)$: $\mathsf{Red}^{\sigma\cap\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] = \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \cap \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

$(\rho \equiv \sigma \rightarrow \tau)$: $\mathsf{Red}^{\sigma\rightarrow\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ is the set of terms $t$ typeable with $(\sigma\rightarrow\tau)[\overline{\gamma}/\overline{\varphi}]$ and such that, for every $u \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, $Ap(t,u) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

$(\rho \equiv \forall\alpha'.\tau)$: $\mathsf{Red}^{\forall\alpha'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ is the set of terms $t$ typeable with $\forall\alpha'.\tau[\overline{\gamma}/\overline{\varphi}]$ and such that, for any type $\delta$ and reducibility candidate $\mathcal{S}^{\delta}$, $t \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\alpha']$.

From now on, when considering a sequence $\overline{\mathcal{R}^{\gamma}}$ in a $\mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, we shall always tacitly assume $|\overline{\mathcal{R}^{\gamma}}| = |\overline{\varphi}|$ and $\{\overline{\varphi}\} \supseteq FV(\rho)$.

*Lemma 3.4* $\mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ *is a reducibility candidate of type* $\rho[\overline{\gamma}/\overline{\varphi}]$.

*Proof:* By induction on the structure of $\rho$.

$(\rho \equiv s)$: By definition $\mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] = \mathcal{SN}^{s}$. It is straightforward to check that $\mathcal{SN}^{s}$ satisfies ($C1$), ($C2$) and ($C3$).

$(\rho \equiv \varphi_i)$: Immediate by definition, since $\mathcal{R}_i^{\gamma_i}$ is a reducibility candidate of type $\gamma_i$.

$(\rho \equiv \sigma\cap\tau)$: We have first to show that, if $t \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \cap \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, then $t$ is typeable with $(\sigma\cap\tau)[\overline{\gamma}/\overline{\varphi}]$. By induction $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ and $\mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ are reducibility candidates of type $\sigma[\overline{\gamma}/\overline{\varphi}]$ and $\tau[\overline{\gamma}/\overline{\varphi}]$, respectively. Hence $t$ is typeable by $\sigma[\overline{\gamma}/\overline{\varphi}]$ and $\tau[\overline{\gamma}/\overline{\varphi}]$. Since $(\sigma\cap\tau)[\overline{\gamma}/\overline{\varphi}] \equiv \sigma[\overline{\gamma}/\overline{\varphi}] \cap \tau[\overline{\gamma}/\overline{\varphi}]$, by definition of our system we get that $t$ is typeable with $(\sigma\cap\tau)[\overline{\gamma}/\overline{\varphi}]$. We have now to prove the other properties of a reducibility candidate, namely ($C1$), ($C2$) and ($C3$). These are easily inferred by the fact that $\mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ is the intersection of two reducibility candidates.

$(\rho \equiv \sigma\rightarrow\tau)$: By definition 3.3 we have that if $t \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ then $t$ is typeable with $(\sigma\rightarrow\tau)[\overline{\gamma}/\overline{\varphi}]$. We can prove now the other properties which must hold for a reducibility candidate.

($C1$): Let $t \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. By induction both $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ and $\mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ are reducibility candidates. Hence, since any reducibility candidate contains all the variables, by Definition 3.3, we have that $Ap(t,x) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ and, by ($C1$), $Ap(t,x)$ is strongly normalisable. Thus also $t$ is strongly normalisable.

($C2$): Let $t \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, with $t \rightarrow^* t'$, and let $u \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. By Definition 3.3, we have $Ap(t,u) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Hence $Ap(t,u) \rightarrow^* Ap(t',u)$ and $Ap(t',u) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, by ($C2$). By Subject Reduction (Theorem 2.27) and ($C2$), we obtain $t' \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

($C3$): Let $t$ be neutral and typeable with $\rho[\overline{\gamma}/\overline{\varphi}]$, and let us assume that

$$\forall u.t \rightarrow u \Rightarrow u \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}].$$

We have to prove that

$$\forall w.(w \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \Rightarrow Ap(t,w) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]).$$

Let then $v \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Since $t$ is typeable with $(\sigma\rightarrow\tau)[\overline{\gamma}/\overline{\varphi}]$ and, by induction, $v$ is typeable with $\sigma[\overline{\gamma}/\overline{\varphi}]$, $Ap(t,v)$ is typeable with $\tau[\overline{\gamma}/\overline{\varphi}]$. Moreover $Ap(t,v)$ is a neutral term and thus, since ($C3$) holds for $\mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ by induction, to prove that $Ap(t,v) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, it suffices to show that for any $\hat{t}$ such that $Ap(t,v) \rightarrow \hat{t}$, $\hat{t} \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. By ($C1$) for $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, $v$ is strongly normalisable. We now proceed by induction on the height of the reduction tree of $v$.

Since $t$ is neutral, the reduction from $Ap(t,v)$ to $\hat{t}$ has necessarily to occur either in $t$ or in $v$.

1) If $\hat{t} \equiv Ap(t',v)$ with $t \to t'$ then, by our assumption $t' \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ and hence, by definition of $\mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ we have $Ap(t',v) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

2) If $\hat{t} \equiv Ap(t,v')$ with $v \to v'$, then, by (*C2*) for $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ we have $v' \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Since the reduction tree of $v'$ is strictly shorter than the one of $v$, by induction we obtain $Ap(t,v') \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

$(\rho \equiv \forall \varphi'.\tau)$: Any element of $\mathsf{Red}^{\forall \varphi'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ is typeable with $\forall \varphi'.\tau[\overline{\gamma}/\overline{\varphi}]$ by definition. We now check that the other conditions hold.

(*C1*): Let $t \in \mathsf{Red}^{\forall \varphi'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, let $\delta$ be an arbitrary type and let $\mathcal{S}^{\delta}$ be a reducibility candidate for $\delta$ (the strongly normalisable terms typeable with $\delta$, for instance). Then, by definition, $t \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$. Since for $\mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$ the induction hypothesis applies, by (*C1*) we get that $t$ is strongly normalisable.

(*C2*): Let $t \in \mathsf{Red}^{\forall \varphi'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ with $t \to^{*} t'$. By definition, for all types $\delta$ and candidates $\mathcal{S}^{\delta}$ we have $t \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$. Hence, by induction and (C2), $t' \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$ for all types $\delta$ and candidates $\mathcal{S}^{\delta}$. Thus, by definition and Subject Reduction, $t' \in \mathsf{Red}^{\forall \varphi'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

(*C3*): Let $t$ be neutral and typeable with $\forall \varphi'.\tau[\overline{\gamma}/\overline{\varphi}]$ and let us assume

$$\forall u.t \to u \;\Rightarrow\; u \in \mathsf{Red}^{\forall \varphi'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}].$$

Taking any type $\delta$ and any candidate $\mathcal{S}^{\delta}$ for it, we have, by definition, that

$$\forall u.t \to u \;\Rightarrow\; u \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi'].$$

Since the induction hypothesis applies for $\mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$, by (*C3*) we have that, for any type $\delta$ and any candidate $\mathcal{S}^{\delta}$ for it, $t \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$, and hence, by definition, $t \in \mathsf{Red}^{\forall \varphi'.\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. ■

*Lemma 3.5* (Red-SUBSTITUTION LEMMA)

$$\mathsf{Red}^{\sigma[\tau/\varphi]}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \;\equiv\; \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathsf{Red}^{\tau/\varphi}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]].$$

*Proof:* By induction on the structure of $\sigma$. ■

*Lemma 3.6 Let $\tau \leq \sigma$. Then, for any reducibility candidates $\overline{\mathcal{R}^{\gamma}}$:*

$$\mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \;\subseteq\; \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}].$$

*Proof:* By induction on the definition of $\leq$.

$(\forall 1 \leq i \leq n \ (n \geq 1)\ [\sigma_1 \cap \cdots \cap \sigma_n \leq \sigma_i])$: Easy.

$(\forall 1 \leq i \leq n \ (n \geq 1) \sigma \leq \sigma_i \;\Rightarrow\; \sigma \leq \sigma_1 \cap \cdots \cap \sigma_n)$: Easy.

$(\sigma \leq \sigma' \;\&\; \tau \leq \tau' \;\Rightarrow\; \sigma' \to \tau \leq \sigma \to \tau')$: Let $t \in \mathsf{Red}^{\sigma' \to \tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Then, by definition, $t$ is typeable with $\sigma' \to \tau$ and

$$\forall u.(u \in \mathsf{Red}^{\sigma'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \;\Rightarrow\; Ap(t,u) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]).$$

In order to prove that $t \in \mathsf{Red}^{\sigma \to \tau'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ we first notice that, by Theorem 2.18, $t$ is also typeable with $\sigma \to \tau'$. Now, to show that

$$\forall u.(u \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \;\Rightarrow\; Ap(t,u) \in \mathsf{Red}^{\tau'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]),$$

take $u \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. By induction $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \subseteq \mathsf{Red}^{\sigma'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Hence $u \in \mathsf{Red}^{\sigma'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ and by assumption $Ap(t,u) \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Again by induction, $Ap(t,u) \in \mathsf{Red}^{\tau'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

($\forall\varphi'.\sigma \leq \sigma[\tau/\varphi']$): Let $t \in \mathsf{Red}^{\forall\varphi'.\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$. By definition, for any type $\delta$ and reducibility candidate $\mathcal{S}^\delta$, $t \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathcal{S}^\delta/\varphi']$. In particular, $t \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathsf{Red}^{\tau}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]/\varphi']$. By the Red-substitution Lemma 3.5, we obtain $t \in \mathsf{Red}^{\sigma[\tau/\varphi']}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$.

($\sigma \leq \tau \Rightarrow \forall\varphi'.\sigma \leq \forall\varphi'\tau$): By induction, for any reducibility candidates $\overline{\mathcal{R}^\gamma}$, $\mathcal{S}^\delta$:

$$\mathsf{Red}^{\tau}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathcal{S}^\delta/\varphi'] \subseteq \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathcal{S}^\delta/\varphi'].$$

Then, by Definition 3.3, it follows that for any reducibility candidate $\overline{\mathcal{R}^\gamma}$

$$\mathsf{Red}^{\forall\varphi'.\tau}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}] \subseteq \mathsf{Red}^{\forall\varphi'.\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}].$$

($\sigma \leq \forall\varphi'.\sigma$ *with* $\varphi' \notin FV(\sigma)$): Since $\varphi' \notin FV(\sigma)$, for any type $\delta$ and candidate $\mathcal{S}^\delta$, $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}] \equiv \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathcal{S}^\delta/\varphi']$. Hence, by definition, we have that $\mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}] \subseteq \mathsf{Red}^{\forall\varphi.\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$.

**Definition 3.7** *i*) A term $t$ typeable with type $\tau$ is *reducible* if it is in $\mathsf{Red}^{\tau}[\overline{\mathcal{SN}^X}/\overline{X}]$, where $\overline{X}$ are the free variables of $\tau$.

*ii*) Let $B = \{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\}$ be a basis and let $FV(\sigma_1,\ldots,\sigma_n) \subseteq \{\overline{\varphi}\}$. Moreover, let $\overline{\gamma}$ be a sequence of types such that $|\overline{\gamma}| = |\overline{\varphi}|$ and let $\overline{\mathcal{R}^\gamma}$ be a sequence of reducibility candidates. A term-substitution R is $\overline{\mathcal{R}^\gamma}$*-reducible for* $B$ if, for every $x_i{:}\sigma_i \in B$, $x_i^{\mathrm{R}} \in \mathsf{Red}^{\sigma_i}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$.

Terms can, as usual, be seen as trees; the sub-term of $t$ at position $p$ will be denoted by $t|_p$, and $t[u]_p$ will denote the result of replacing, in $t$, the sub-term at position $p$ by $u$.

We shall prove our strong normalisation result by showing that every typeable term is reducible. This implies strong normalisation by Lemma 3.4 and (*C1*). In order to show that any typeable term is reducible we need to prove a stronger property, for which we will need the following ordering.

**Definition 3.8** *i*) Let '$>_{\mathbb{N}}$' denote the standard ordering on natural numbers, and *lex*, *mul* denote respectively the *lexicographic* and *multi-set* extension of an ordering. Let '$\trianglerighteq$' stand for the well-founded encompassment ordering, i.e. $u \trianglerighteq v$ if $u \not\equiv v$ modulo renaming of variables, and $u|_p = v^{\mathrm{R}}$ for some position $p \in u$ and term-substitution R. Note that encompassment contains strict super-term ('$\triangleright$').

*ii*) We define the ordering '$\gg^{SN}$' on triples – consisting of a natural number, a term, and a multi-set of terms – as the object

$$(>_{\mathbb{N}}, \trianglerighteq, (\rightarrow \cup \triangleright)_{mul})_{lex}.$$

*iii*) We will interpret the term $u^{\mathrm{R}}$ by the triple $\langle i, u, \{\mathrm{R}\}\rangle = \mathcal{I}(u^{\mathrm{R}})$, where

- $i$ is the maximal super-index of the function symbols belonging to $u$,
- $\{\mathrm{R}\}$ is the multi-set $\{x^{\mathrm{R}} \mid x \in Var(u)\}$.

These triples are compared in the ordering '$\gg^{SN}$'.

When R is $\overline{\mathcal{R}^\gamma}$-reducible for some basis containing the free variables of $u$, then, by (*C1*), every $t$ in $\{\mathrm{R}\}$ is strongly normalisable, so the rewrite relation '$\rightarrow$' is well-founded on $\{\mathrm{R}\}$. Also, since the union of the relation '$\triangleright$' with a terminating rewrite relation is well-founded [18], the relation '$(\rightarrow \cup \triangleright)_{mul}$' is well-founded on $\{\mathrm{R}\}$. Hence, with such term-substitutions, '$\gg^{SN}$' is a well-founded ordering.

We will use '$\gg^{SN}_n$' when we want to indicate that the $n$-th element of the triple has decreased and the first $n-1$ have not increased.

We would like to stress that we do not just interpret terms, but pairs of terms and term-substitutions. This implies that although it can be that the terms $t^{\mathrm{R}_1}$ and $t^{\mathrm{R}_2}$ are equal, their interpretations need not be equal as well.

We now come to the main theorem of this section.

*Property 3.9* *Let $B \vdash_{\mathcal{E}} t{:}\sigma$, where $B = \{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\}$, and let $FV(\sigma_1,\ldots,\sigma_n,\sigma) \subseteq \{\overline{\varphi}\}$. Then, given a sequence of types $\overline{\gamma}$ such that $|\overline{\gamma}| = |\overline{\varphi}|$ and a sequence of reducibility candidates $\overline{\mathcal{R}^{\gamma}}$, if* R *is a term-substitution that is $\overline{\mathcal{R}^{\gamma}}$-reducible for $B$, then $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.*

*Proof:* See the Appendix.

**Theorem 3.10** (Strong Normalisation) *Any typeable term is strongly normalisable.*

*Proof:* By Property 3.9, it easily follows that any typeable term is reducible. Strong normalisation then follows from Lemma 3.4 and (*C1*). ■

## 4 Confluence

Using the previous strong normalisation result, we are going to show that the absence of critical pairs in a typeable TRS + $\beta$ implies confluence on typeable terms.

**Definition 4.1** If $l \to r$ and $s \to t$ are two rewrite rules (we assume that the variables were renamed so that there is no variable that occurs in both), $p$ is the position of a non-variable subterm of $s$ and $\mu$ is a most general unifier of $s|_p$ and $l$, then $(t\mu, s\mu[r\mu]_p)$ is a *critical pair* formed from those rules. Note that the second rule may be a renamed version of the first. In this case a super-position at the root position is not considered a critical pair.

We will prove that the absence of critical pairs implies *local confluence*, and use Newman's Lemma [33] to deduce confluence from strong normalisation and local confluence. Let us recall the definition of local confluence:

**Definition 4.2** A reduction relation $\to$ is *locally confluent* on a set $T$ of terms, if for any $t, v_1, v_2 \in T$ such that $t \to v_1$ and $t \to v_2$, there exists $v_3 \in T$ such that $v_1 \to^* v_3$ and $v_2 \to^* v_3$.

**Theorem 4.3** *A typeable* TRS + $\beta$ $(\Sigma, \mathbf{R})$ *is locally confluent on typeable terms if it does not have critical pairs.*

By the Subject Reduction Theorem, all rewrite sequences starting from a typeable term remain inside the set of typeable terms. We study the interactions between the two classes of reductions we have: $\beta$-reductions, and reductions using $\mathbf{R}$.
The absence of critical pairs guarantees no super-position between the rewrite rules. It is well-known that this implies local confluence of first-order rules on algebraic terms. The extension to terms containing $\lambda$-abstractions is standard: we abstract with term-variables the sub-terms that have a $\lambda$ at the root, taking care of using the same variable for identical sub-terms since rewrite rules may be non left-linear (see [9] for details). The extension to rules containing $\lambda$-abstraction only in the right-hand side (as in our systems) is also straightforward.
Since $\beta$-reductions are confluent on $\lambda$-terms containing constants, the only remaining case to study is the interaction of $\beta$-reductions with reductions using $\mathbf{R}$. But since by definition of rewrite rule, the symbol $Ap$ cannot appear in a left-hand side, there is no super-position between $\beta$ and other rules. This completes the proof. ■

## 5 Restriction to Rank2

In this section, we will present a decidable restriction of the type system as presented above, based on types of rank 2. Although the Rank 2 intersection system and the Rank 2 polymorphic system for LC type exactly the same set of terms [37], their combination results in a system with more expressive power: polymorphism can be expressed directly (using the universal quantifier) and, more importantly, as we will show below, every typeable term has a principal type; the latter property does not hold in a system without intersection.

## 5.1 Rank 2 type assignment

In this subsection, we will briefly discuss *a* notion of Rank 2 type assignment (the system presented here is not the only one possible: a variant could be to consider also the empty intersection, i.e. to use the type constant $\omega$, but we will not take that direction here).

The polymorphic intersection types of Rank 2, $\mathcal{T}_2$, are a true subset of the set of polymorphic intersection types as presented in Definition 2.1.

**Definition 5.1** (Rank 2 types and bases) *i*) We define polymorphic intersection types of Rank 2 in layers:

$$\begin{array}{rcll} \mathcal{T}_C & ::= & \varphi \mid s \mid (\mathcal{T}_C \rightarrow \mathcal{T}_C) & \text{(Curry types)} \\ \mathcal{T}^{\forall}{}_C & ::= & \mathcal{T}_C \mid (\forall \alpha.\mathcal{T}^{\forall}{}_C[\alpha/\varphi]) & \text{(Quantified Curry types)} \\ \mathcal{T}_1 & ::= & (\mathcal{T}^{\forall}{}_C \cap \cdots \cap \mathcal{T}^{\forall}{}_C) & \text{(Rank 1 types)} \\ \mathcal{T}_2 & ::= & \varphi \mid (\mathcal{T}_1 \rightarrow \mathcal{T}_2) & \text{(Rank 2 types)} \end{array}$$

We omit brackets as before.

*ii*) A *Rank 2 basis* is a basis in which all types are in $\mathcal{T}_1$.

Below, we will define a unification procedure that will recursively go through types. However, using the sets defined above, not every sub-type of a type in $\mathcal{T}_2$ is a type in that same set. For example, $\alpha \rightarrow \varphi$ is not a type in any of the sets defined above; however, $\forall \alpha.\alpha \rightarrow \varphi \in \mathcal{T}^{\forall}{}_C$, and therefore it can be that, when going through types in $\mathcal{T}_2$ recursively, $\alpha \rightarrow \varphi$ has to be dealt with. Since the distinction between free and bound variables is essential, we introduce, for every set $\mathcal{T}_i$ defined above, also the set $\mathcal{T}_i'$ of types, that contains also free occurrences of $\alpha$s. We will not always use the '$'$' when speaking of these sets, however; it will be clear from the context which set is intended.

As for $\mathcal{T}$, we will consider a relation on types, $\leq_2$, but one that is *not* the restriction to $\mathcal{T}_2$ of the relation $\leq$ defined in Definition 2.2; notice that the part corresponding to '$\sigma \leq_2 \forall \alpha.\sigma$, if $\alpha$ not in $\sigma$' is missing.

**Definition 5.2** (Relations on types) On types, the pre-order (i.e. reflexive and transitive relation) $\leq_2$ is generated by the following rules:

$$\begin{array}{rcll} \sigma_1 \cap \cdots \cap \sigma_n & \leq_2 & \sigma_i, & (1 \leq i \leq n) \\ \forall \alpha.\sigma[\alpha/\varphi] & \leq_2 & \sigma[\tau/\varphi], & (\tau \in \mathcal{T}_C) \\ \forall 1 \leq i \leq n.\sigma \leq_2 \sigma_i & \Rightarrow & \sigma \leq_2 \sigma_1 \cap \cdots \cap \sigma_n & (n \geq 1) \\ \rho \leq_2 \sigma, \tau \leq_2 \mu & \Rightarrow & \sigma \rightarrow \tau \leq_2 \rho \rightarrow \mu, & (\rho, \sigma \in \mathcal{T}_1,\ \tau, \mu \in \mathcal{T}_2) \\ \sigma \leq_2 \tau & \Rightarrow & \forall \alpha.\sigma[\alpha/\varphi] \leq_2 \forall \alpha.\tau[\alpha/\varphi]. & \end{array}$$

The equivalence relation '$\sim_2$' is defined by: $\sigma \sim_2 \tau \iff \sigma \leq_2 \tau \leq_2 \sigma$, and we extend '$\leq_2$' to bases in the same way as done for '$\leq$'.

For $\leq_2$, the following properties hold:

*Lemma 5.3 i) If $\sigma \in \mathcal{T}_1$, $\sigma \leq_2 \tau \in \mathcal{T}_2$, and $\sigma$ does not contain '$\forall$', then neither does $\tau$.*

*ii) If $\sigma \leq_2 \tau_1 \cap \cdots \cap \tau_n$, then, for all $1 \leq i \leq n$, $\sigma \leq_2 \tau_i$.*

*Proof:* Easy. ■

The Rank 2 versions for the various operations as presented below are defined in much the same way as in [6], with the exception of the operation of closure and lifting, that were not used there.

The first three operations used for the Rank 2 system are straightforward variants of operations defined for the full system.

**Definition 5.4** *i*) *Substitution* $(\varphi \mapsto \rho) : \mathcal{T}_2 \to \mathcal{T}_2$ is defined as in Definition 2.5, but with the restriction that $\rho \in \mathcal{T}_C$. For the sake of clarity, and in order to avoid writing $[S_1,\ldots,S_n]$ for a chain of single type-variable substitutions, we will close the set of substitution for composition '$\circ$': for substitutions $S_1,S_2$, the substitution $S_2{\circ}S_1$ is defined as $S_2{\circ}S_1(\sigma) = S_2(S_1(\sigma))$. We use $Id_S$ for the substitution that replaces all variables by themselves, and write $\mathcal{S}$ for the set of all substitutions.

*ii*) *Lifting* is defined as in Definition 2.12, but with the restriction that '$\leq$' is taken to be '$\leq_2$' of Definition 5.2.

*iii*) *Closure* is defined as in Definition 2.13 as a pair of types $\langle \sigma,\varphi \rangle$, with the restriction that $\sigma \in \mathcal{T}^{\forall}{}_C$:

$$\langle \sigma,\varphi \rangle (\langle B,\tau_1{\cap}\cdots{\cap}\tau_n \rangle) = \langle B,\tau'_1{\cap}\cdots{\cap}\tau'_n \rangle$$

where, for all $1 \leq i \leq n$,

$$\begin{array}{lll} \tau'_i & = \forall\alpha.\sigma[\alpha/\varphi], & \text{if } \tau_i = \sigma, \text{ and } \varphi \text{ does not appear in } B \text{ ($\alpha$ is a fresh variable), and} \\ \tau'_i & = \tau_i, & \text{otherwise.} \end{array}$$

The variant of expansion used in the Rank 2 system is quite different from that of Definition 2.9. The reason for this is that expansion, normally, increases the rank of a type:

$$\langle \varphi_1,2 \rangle (\langle \{x{:}\varphi_1{\to}\varphi_2\},\varphi_1 \rangle)(\varphi_1{\to}\varphi_2) = (\varphi_1^1{\cap}\varphi_1^2){\to}\varphi_2,$$

a feature that is of course not sound when present within a system that limits the rank of types. Since below expansion is only used in very precise situations (within the procedure *unify*$^{\forall}{}_2$, and in the proof of Theorem 5.29), the solution is relatively easy: in the context of Rank 2 types, expansion is only called on types in $\mathcal{T}^{\forall}{}_C$, so it is defined to work well there, by replacing *all* types by an intersection; in particular, intersections are not created at the left of an arrow.

**Definition 5.5** Let $B$ be a Rank 2 basis, $\sigma \in \mathcal{T}_2$, and $n \geq 1$. The $n$-fold *Rank 2 expansion* with respect to the pair $\langle B,\sigma \rangle$, $n_{\langle B,\sigma \rangle} : \mathcal{T}_2 \to \mathcal{T}_2$ is constructed as follows: Suppose $V = \{\varphi_1,\ldots,\varphi_m\}$ is the set of all (free) variables occurring in $\langle B,\sigma \rangle$. Choose $m \times n$ different variables $\varphi_1^1,\ldots,\varphi_1^n$, $\ldots$, $\varphi_m^1,\ldots,\varphi_m^n$, such that each $\varphi_j^i$ $(1 \leq i \leq n,\ 1 \leq j \leq m)$ does not occur in $V$. Let $S_i$ be the substitution that replaces every $\varphi_j$ by $\varphi_j^i$. Then Rank 2 expansion is defined on types, bases, and pairs, respectively, by:

$$\begin{array}{ll} n_{\langle B,\sigma \rangle}(\tau) & = S_1(\tau) \cap \cdots \cap S_n(\tau), \\ n_{\langle B,\sigma \rangle}(B') & = \{x{:}n_{\langle B,\sigma \rangle}(\rho) \mid x{:}\rho \in B\}, \\ n_{\langle B,\sigma \rangle}(\langle B',\sigma' \rangle) & = \langle n_{\langle B,\sigma \rangle}(B'),n_{\langle B,\sigma \rangle}(\sigma') \rangle. \end{array}$$

Notice that, if $\tau \in \mathcal{T}_2$, it can be that $S_1(\tau) \cap \cdots \cap S_n(\tau)$ is not a legal type. However, since each $S_i(\tau) \in \mathcal{T}_2$, for $1 \leq i \leq n$, for the sake of clarity, we will not treat it separately (see also Lemma 5.13).

Notice that we have no need for the third parameter '$A$' in this notion of expansion. Since Rank 2 expansion essentially is just the combination of a number of substitutions by means of rule $(\cap I)$, we do not need to calculate the set of affected types, for which the third parameter was added.

Since all results in this section regard the Rank 2 system, we will use 'expansion' rather than 'Rank 2 expansion.'

As before, operations will be grouped in chains.

**Definition 5.6** A *Rank 2 chain* (or chain for short) is a chain $Ch$ of operations, composed of at most one expansion, at most one substitution, at most one lifting, and a number ($\geq 0$) of closures:

$$Ch = [Ex,S,L,Cl_1,\ldots Cl_m] = [Ex,S,L,\overline{Cl}].$$

For chains, the following properties hold:

*Lemma 5.7 Let Ch be a chain.*

*i) If $\sigma \in \mathcal{T}_C$, and $Ch(\sigma) \in \mathcal{T}_2$, then there are a substitution $S$ and lifting $L$ such that $Ch(\sigma) = [S,L](\sigma)$.*

*ii) If $\sigma \in \mathcal{T}_2$, and $Ch(\sigma) \in \mathcal{T}^{\forall}{}_C$, then $Ch(\sigma) = [S,Cl_1,\ldots,Cl_n](\sigma)$ for some substitution $S$ and closures $Cl_1, \ldots, Cl_n$.*

*iii) If $\sigma \in \mathcal{T}_C$, and $Ch(\sigma) \in \mathcal{T}_1$, then there exists a lifting-free chain $Ch'$ such that $Ch(\sigma) = Ch'(\sigma)$.*

*iv) If $\sigma \in \mathcal{T}_2$, and $Ch(\sigma) \in \mathcal{T}_C$, then $Ch(\sigma) = S(\sigma)$ for some substitution $S$.*

*v) If $\sigma \in \mathcal{T}_2$, and $Ch(\sigma) \in \mathcal{T}_2$, then $Ch(\sigma) = [S,L](\sigma)$ for some substitution $S$ and lifting $L$.*

*Proof:* For part one, clearly expansion and closure are not needed, and by Lemma 5.3**??**, neither is lifting. The other parts are just generalisations of the first. ■

We now come to the definition of Rank 2 type assignment.

**Definition 5.8** *i*) A *Rank 2 environment* $\mathcal{E}$ is a mapping from $\mathcal{F}$ to $\mathcal{T}_2$.

*ii*) *Rank 2 type assignment on terms* is defined by the following natural deduction system:

$$(\leq_2): \frac{}{B \vdash^2_{\mathcal{E}} x{:}\tau}\,(a) \qquad (\cap I): \frac{B \vdash^2_{\mathcal{E}} t{:}\sigma_1 \quad \cdots \quad B \vdash^2_{\mathcal{E}} t{:}\sigma_n}{B \vdash^2_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n}\,(b)$$

$$(\rightarrow I): \frac{B, x{:}\sigma \vdash^2_{\mathcal{E}} t{:}\tau}{B \vdash^2_{\mathcal{E}} \lambda x.t{:}\sigma\rightarrow\tau} \qquad (\rightarrow E): \frac{B \vdash^2_{\mathcal{E}} t_1{:}\sigma\rightarrow\tau \quad B \vdash^2_{\mathcal{E}} t_2{:}\sigma}{B \vdash^2_{\mathcal{E}} Ap(t_1,t_2){:}\tau}$$

$$(\forall I): \frac{B \vdash^2_{\mathcal{E}} t{:}\sigma}{B \vdash^2_{\mathcal{E}} t{:}\forall\alpha.\sigma[\alpha/\varphi]}\,(c) \qquad (\mathcal{F}): \frac{B \vdash^2_{\mathcal{E}} t{:}\sigma_1 \quad \cdots \quad B \vdash^2_{\mathcal{E}} t{:}\sigma_n}{B \vdash^2_{\mathcal{E}} F(t_1,\ldots,t_n){:}\sigma}\,(d)$$

*a*) If $\sigma \leq_2 \tau$ , $\sigma \in \mathcal{T}_1$, and $\tau \in \mathcal{T}_2$.

*b*) If $n \geq 1$, and $\sigma_i \in \mathcal{T}^{\forall}{}_C$, for every $1 \leq i \leq n$.

*c*) If $\varphi$ does not occur in $B$, and $\sigma \in T^{\forall}_C$.

*d*) If $F \in \mathcal{F}$, and there exists a chain $Ch$ such that $\sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\sigma = Ch(\mathcal{E}(F))$.

Notice that, since quantification elimination is implicit in rule $(\leq_2)$, when restricting the use of the quantifier to the left of arrows only, there is no longer need for a general $(\forall E)$ rule; as rule $(\cap E)$, its use is in a strict system limited to variables, and there its actions are already performed by $(\leq_2)$. In fact, this change is justified by Lemma 5.11.

*Example 5.9* We can derive both

$$\dfrac{\dfrac{}{\{y{:}(\sigma\to\tau)\cap\sigma\} \vdash^2_{\mathcal{E}} y:\sigma\to\tau}\,(\leq_2) \qquad \dfrac{}{\{y{:}(\sigma\to\tau)\cap\sigma\} \vdash^2_{\mathcal{E}} y:\sigma}\,(\leq_2)}{\dfrac{\{y{:}(\sigma\to\tau)\cap\sigma\} \vdash^2_{\mathcal{E}} yy:\tau}{\emptyset \vdash^2_{\mathcal{E}} \lambda y.yy:(\sigma\to\tau)\cap\sigma\to\tau}}$$

and

$$\dfrac{\dfrac{}{\{y{:}\forall\alpha.\alpha\} \vdash^2_{\mathcal{E}} y:\sigma\to\tau}\,(\leq_2) \qquad \dfrac{}{\{y{:}\forall\alpha.\alpha\} \vdash^2_{\mathcal{E}} y:\sigma}\,(\leq_2)}{\dfrac{\{y{:}\forall\alpha.\alpha\} \vdash^2_{\mathcal{E}} yy:\tau}{\emptyset \vdash^2_{\mathcal{E}} \lambda y.yy:\forall\alpha.\alpha\to\tau}}$$

We will now show that the above defined operations are sound. First, we show this for substitution.

*Lemma 5.10* *If $B \vdash^2_{\mathcal{E}} t:\sigma$, and $S$ is a substitution, then $S(B) \vdash^2_{\mathcal{E}} t:S(\sigma)$.*

*Proof:* By induction on the structure of derivations.

$(\leq_2)$: Then $t = x, \sigma \in \mathcal{T}_2$, and there is $\rho \in \mathcal{T}_1$ such that $x{:}\rho \in B$ and $\rho \leq_2 \sigma$. Since $S(\rho) \leq_2 S(\sigma)$, and $x{:}S(\rho) \in S(B)$, also $S(B) \vdash^2_{\mathcal{E}} x:S(\sigma)$.

$(\cap I)$: Then $\sigma = \sigma_1\cap\cdots\cap\sigma_n$, and, for $1 \leq i \leq n$, $B \vdash^2_{\mathcal{E}} t:\sigma_i$. Then, by induction, for all $1 \leq j \leq m$, $S(B) \vdash^2_{\mathcal{E}} t:S(\tau_j)$, so, by rule $(\cap I)$, also $S(B) \vdash^2_{\mathcal{E}} t:S(\tau_1\cap\cdots\cap\tau_m)$.

$(\to I)$: Then $t = \lambda x.t'$, and there are $\rho, \mu$ such that $\sigma = \rho\to\mu$ and $B, x{:}\rho \vdash^2_{\mathcal{E}} t':\mu$. By induction, $S(B), x{:}S(\rho) \vdash^2_{\mathcal{E}} t':S(\mu)$, so, by rule $(\to I)$, $S(B) \vdash^2_{\mathcal{E}} \lambda x.t':S(\rho\to\mu)$.

$(\to E)$: Then $t = Ap(t_1,t_2)$, and there is a $\rho \in \mathcal{T}_1$ such that $B \vdash^2_{\mathcal{E}} t_1:\rho\to\sigma$, and $B \vdash^2_{\mathcal{E}} t_2:\rho$. By induction, $S(B) \vdash^2_{\mathcal{E}} t_1:S(\rho\to\sigma)$, and $S(B) \vdash^2_{\mathcal{E}} t_2:S(\rho)$, so, by rule $(\to E)$, $S(B) \vdash^2_{\mathcal{E}} Ap(t_1,t_2):S(\sigma)$.

$(\forall I)$: Then $\sigma = \forall\alpha.\rho[\alpha/\varphi]$, and $B \vdash^2_{\mathcal{E}} t:\rho$. By induction, $S(B) \vdash^2_{\mathcal{E}} t:S(\rho)$. We can assume, without loss of generality, that $\varphi$ is not affected by $S$, so, $\varphi$ occurs in $\rho$ if and only if it occurs in $S(\rho)$. Therefore, by rule $(\forall I)$, also $S(B) \vdash^2_{\mathcal{E}} t:S(\rho)[\alpha/\varphi]$, so $S(B) \vdash^2_{\mathcal{E}} t:S(\rho[\alpha/\varphi])$.

$(\mathcal{F})$: Then $t = F(t_1,\ldots,t_n)$, and there are $\sigma_1,\ldots,\sigma_n$ such that, for $1 \leq i \leq n$, $B \vdash^2_{\mathcal{E}} t_i:\sigma_i$, and there exists a chain $Ch$ such that $Ch(\mathcal{E}(F)) = \sigma_1\to\cdots\to\sigma_n\to\sigma$. By induction, for $1 \leq i \leq n$, $S(B) \vdash^2_{\mathcal{E}} t_i:S(\sigma_i)$, and since

$$[S] * Ch(\mathcal{E}(F)) = S(\sigma_1)\to\cdots S(\sigma_n)\to S(\sigma),$$

by rule $(\mathcal{F})$ we obtain $S(B) \vdash^2_{\mathcal{E}} F(t_1,\ldots,t_n):S(\sigma)$. ∎

The next lemma states essentially that lifting is a sound operation.

*Lemma 5.11* *If $B \vdash^2_{\mathcal{E}} t:\sigma$, and let $B', \tau$ be such that $B' \leq_2 B$, and $\sigma \leq_2 \tau$, then $B' \vdash^2_{\mathcal{E}} t:\tau$.*

*Proof:* By induction on the structure of derivations. First we deal with the case that $\tau$ is not an intersection.

$(\leq_2)$: Then $t = x, \sigma \in \mathcal{T}_2$, and there is $\rho \in \mathcal{T}_1$ such that $x{:}\rho \in B$ and $\rho \leq_2 \sigma$. Since $B' \leq_2 B$, there is $\rho' \in \mathcal{T}_1$ such that $x{:}\rho' \in B'$ and $\rho' \leq_2 \rho \leq_2 \sigma$. Since $\sigma \leq_2 \tau$, also $\rho' \leq_2 \tau$ and $B' \vdash^2_{\mathcal{E}} x:\tau$. (Notice that, since $\tau$ is not an intersection, $\tau \in \mathcal{T}_2$.)

$(\cap I)$: Then $\sigma = \sigma_1\cap\cdots\cap\sigma_n$, and, for $1 \leq i \leq n$, $B \vdash^2_{\mathcal{E}} t:\sigma_i$. Then there is an $1 \leq i \leq n$, such that $\sigma_i \leq_2 \tau$. Then, by induction, $B' \vdash^2_{\mathcal{E}} t:\tau$.

$(\rightarrow I)$: Then $t = \lambda x.t'$, and there are $\rho,\mu$ such that $\sigma = \rho\rightarrow\mu$ and $B,x{:}\rho \vdash^2_{\mathcal{E}} t'{:}\mu$, and $\delta,\gamma$ such that $\tau = \gamma\rightarrow\delta$, and $\gamma \leq_2 \rho, \mu \leq_2 \delta$. Then, by induction $B',x{:}\gamma \vdash^2_{\mathcal{E}} t'{:}\delta$, and therefore, by rule $(\rightarrow I)$, also $B' \vdash^2_{\mathcal{E}} \lambda x.t'{:}\gamma\rightarrow\delta$.

$(\rightarrow E)$: Then $t = Ap(t_1,t_2)$, and there is a $\rho \in \mathcal{T}_1$ such that $B \vdash^2_{\mathcal{E}} t_1{:}\rho\rightarrow\sigma$, and $B \vdash^2_{\mathcal{E}} t_2{:}\rho$. Since $\sigma \leq_2 \tau$, also $\rho\rightarrow\sigma \leq_2 \rho\rightarrow\tau$, so by induction, $B' \vdash^2_{\mathcal{E}} t_1{:}\rho\rightarrow\tau$ and, by rule $(\rightarrow E)$, also $B' \vdash^2_{\mathcal{E}} Ap(t_1,t_2){:}\tau$.

$(\forall I)$: Then $\sigma = \forall\alpha.\rho[\alpha/\varphi]$, and $B \vdash^2_{\mathcal{E}} t{:}\rho$. Since $\forall\alpha.\rho[\alpha/\varphi] \leq_2 \tau$, by definition of '$\leq_2$', either:

$(\tau = \rho[\mu/\varphi]$, *with* $\mu \in \mathcal{T}_C)$: By induction, $B' \vdash^2_{\mathcal{E}} t{:}\rho$, and, by Lemma 5.10, $B' \vdash^2_{\mathcal{E}} t{:}\rho[\mu/\varphi]$ (notice that $\varphi$ occurs in $\rho$ only).

$(\tau = \forall\alpha.\mu[\alpha/\varphi]$, *with* $\rho \leq_2 \mu)$: Then, by induction, $B' \vdash^2_{\mathcal{E}} t{:}\mu$, and $B' \vdash^2_{\mathcal{E}} t{:}\forall\alpha.\mu[\alpha/\varphi]$ by rule $(\forall I)$.

$(\mathcal{F})$: Then $t = F(t_1,\ldots,t_n)$, and there are $\sigma_1,\ldots,\sigma_n$ such that, for $1 \leq i \leq n$, $B \vdash^2_{\mathcal{E}} t_i{:}\sigma_i$, and there exists a chain $Ch$ such that $Ch(\mathcal{E}(F)) = \sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\sigma$. Since $B' \leq_2 B$, by induction $B' \vdash^2_{\mathcal{E}} t_i{:}\sigma_i$, for $1 \leq i \leq n$. Since $\sigma \leq_2 \tau$,

$$L = <\langle\emptyset,\sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\sigma\rangle,\langle\emptyset,\sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\tau\rangle>$$

is a lifting, so $[L] * Ch(\mathcal{E}(F)) = \sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\tau$, and $B' \vdash^2_{\mathcal{E}} F(t_1,\ldots,t_n){:}\tau$ by rule $(\mathcal{F})$.

If $\tau = \tau_1\cap\cdots\cap\tau_n$, with each $\tau_i \in \mathcal{T}^{\forall}{}_C$, then, by Lemma 5.3(*ii*), for all $1 \leq i \leq n$, $\sigma \leq_2 \tau_i$. The result then follows from the above, and rule $(\cap I)$. ■

The next lemma states that closure is a sound operation.

*Lemma 5.12 If* $B \vdash^2_{\mathcal{E}} t{:}\tau$ *and let* $Cl = \langle\sigma,\varphi\rangle$ *be a closure such that* $Cl(\langle B,\tau\rangle) = \langle B',\rho\rangle$, *then* $B' \vdash^2_{\mathcal{E}} t{:}\rho$.

*Proof:* Let $\tau = \tau_1\cap\cdots\cap\tau_n$ $(n \geq 1)$, then $\langle\sigma,\varphi\rangle(\langle B,\tau_1\cap\cdots\cap\tau_n\rangle) = \langle B,\tau'_1\cap\cdots\cap\tau'_n\rangle$ so $B' = B$ and, for all $1 \leq i \leq n$,

- $\varphi$ occurs in $B$, and $\tau_i = \sigma$, and the result is trivial, or
- $\varphi$ does not occur in $B$, $\tau_i = \sigma$, and $\tau'_i = \forall\alpha.\sigma[\alpha/\varphi]$, and the result follows from rule $(\forall I)$. ■

Since expansion just creates an intersection of types, it could be that the type created is not in $\mathcal{T}_2$, but would be an intersection of types from that set. Therefore, we cannot show a general soundness result. However, we can show the following:

*Lemma 5.13 Let* $Ex$ *be an expansion, and let* $Ex(\sigma) = \sigma_1\cap\cdots\cap\sigma_n$. *If* $B \vdash^2_{\mathcal{E}} t{:}\sigma$, *then, for every* $1 \leq i \leq n$, *there is a* $B'$ *such that* $B' \vdash^2_{\mathcal{E}} t{:}\sigma_i$.

*Proof:* By Definition 5.5, there are substitutions $S_1,\ldots,S_n$ such that $Ex(\sigma) = S_1(\sigma)\cap\cdots\cap S_n(\sigma)$. The result then follows from Lemma 5.10 (notice that $B' = S_i(B)$). ■

In case expansion gets applied to a type in $\mathcal{T}_1$, the result is stronger.

*Lemma 5.14 Let* $Ex$ *be an expansion. If* $\sigma \in \mathcal{T}_1$ *and* $B \vdash^2_{\mathcal{E}} t{:}\sigma$, *then* $Ex(B) \vdash^2_{\mathcal{E}} t{:}Ex(\sigma)$.

*Proof:* By the previous lemma, if $Ex(\sigma) = \sigma_1\cap\cdots\cap\sigma_n$, then, for every $1 \leq i \leq n$, there is a $B'$ such that $B' \vdash^2_{\mathcal{E}} t{:}\sigma_i$. Since $\sigma \in \mathcal{T}_1$, also each $\sigma_i \in \mathcal{T}_1$. Notice that $Ex(B) \leq_2 S_i(B)$, for $1 \leq i \leq n$, so, by Lemma 5.11 and rule $(\cap I)$, we get the result. ■

We have now the following result:

*Lemma 5.15 If* $\sigma \in \mathcal{T}_1$, $B \vdash^2_{\mathcal{E}} t{:}\sigma$, *and* $Ch$ *is a chain such that* $Ch(\langle B,\sigma\rangle) = \langle B',\sigma'\rangle$, *then* $B' \vdash^2_{\mathcal{E}} t{:}\sigma'$.

*Proof:* By lemmas 5.10 to 5.13. ■

The following properties of chains will be used in the proof of Theorem 5.29 below.

*Lemma 5.16* *i) If there exists a chain Ch such that $Ch(\langle P\cup\{x{:}\nu\},\pi\rangle) = \langle B\cup\{x{:}\rho\},\mu\rangle$, where $\pi,\mu \in \mathcal{T}_2$, then there exists a chain $Ch'$ such that $Ch'(\langle P,\nu\rightarrow\pi\rangle) = \langle B,\rho\rightarrow\mu\rangle$.*

*ii) If there exists a chain Ch such that $Ch(\langle P,\pi\rangle) = \langle B,\mu\rangle$, where $\pi,\mu \in \mathcal{T}_2$, then there exists a chain $Ch'$ such that $Ch'(\langle P,\varphi\rightarrow\pi\rangle) = \langle B,\varphi\rightarrow\mu\rangle$, where $\varphi$ is a fresh type variable.*

*Proof:* Straightforward. ■

## 5.2 Unification of Rank 2 Types

In the context of types, unification is a procedure normally used to find a common instance for demanded and provided type for applications, i.e: if $t_1$ has type $\sigma\rightarrow\tau$, and $t_2$ has type $\rho$, then unification looks for a common instance of the types $\sigma$ and $\rho$ such that $Ap(t_1,t_2)$ can be typed properly. The unification algorithm $\mathit{unify}^{\forall}{}_2$ presented in the next definition deals with just that problem. This means that it is not a full unification algorithm for types of Rank 2, but only an algorithm that finds the most general unifying chain for demanded and provided type. It is defined as a natural extension of Robinson's well-known unification algorithm *unify* [35], and can be seen as an extension of the notion of unification as presented in [6], in that it deals with quantification as well.

**Definition 5.17** Unification of Curry types (extended with bound variables and type constants) is defined by:
$\mathit{unify} : \mathcal{T}_C' \times \mathcal{T}_C' \rightarrow \mathcal{S}$.

$$\begin{array}{lll}
\mathit{unify}(\varphi,\tau) & = (\varphi \mapsto \tau), \text{ if } \varphi \text{ does not occur in } \tau, \text{ or } \varphi = \tau \\
\mathit{unify}(\alpha,\alpha) & = \mathit{Id}_S, \\
\mathit{unify}(s,s) & = \mathit{Id}_S, \\
\mathit{unify}(\sigma,\varphi) & = \mathit{unify}(\varphi,\sigma), \\
\mathit{unify}(\sigma\rightarrow\tau,\rho\rightarrow\mu) & = S_2\circ S_1, \text{ where } S_1 = \mathit{unify}(\sigma,\rho), \\
& \phantom{= S_2\circ S_1, \text{ where }} S_2 = \mathit{unify}(S_1(\tau),S_1(\mu)).
\end{array}$$

(Aa usual, all non-specified cases, like $\mathit{unify}(\alpha_1,\alpha_2)$ with $\alpha_1 \neq \alpha_2$, fail.)

It is worthwhile to notice that the operation on types returned by *unify* is not really a substitution, since it allows, e.g., $\varphi \mapsto \alpha$, without keeping track of the binder for $\alpha$. This potentially will create wrong results, since unification can now substitute bound variables in unbound places. Therefore, special care has to be taken before applying a substitution, to guarantee its application to the argument acts as a 'real' substitution.

The following property is well-known, and formulates that *unify* returns the most general unifier for two Curry types, if it exists.

*Property 5.18* ([35]) *For all $\sigma,\tau \in \mathcal{T}_C$, substitutions $S_1,S_2$: if $S_1(\sigma) = S_2(\tau)$, then there are substitutions $S_u$ and $S'$ such that*

$$S_u = \mathit{unify}(\sigma,\tau), \text{ and } S_1(\sigma) = S'\circ S_u(\sigma) = S'\circ S_u(\tau) = S_2(\tau).$$

The unification algorithm $\mathit{unify}^{\forall}{}_2$ as defined below gets, typically, called during the computation of the principal pair for an application $Ap(t_1,t_2)$. Suppose the algorithm has derived $P_1 \vdash^2_{\mathcal{E}} t_1{:}\pi_1$ and $P_2 \vdash^2_{\mathcal{E}} t_2{:}\pi_2$ as principal pairs for $t_1$ and $t_2$, respectively, and that $\pi_1 = \sigma\rightarrow\tau$. Thus the demanded type $\sigma$ is in $\mathcal{T}_1$ and the provided type $\pi_2$ is in $\mathcal{T}_2$. In order to be consistent, the result of the unification of $\sigma$ and $\pi_2$ – a chain $Ch$ – should always be such that

$Ch(\pi_2) \in \mathcal{T}_1$. However, if $\pi_2 \notin \mathcal{T}_C$, then in general $Ch(\pi_2) \notin \mathcal{T}_1$. To overcome this difficulty, an algorithm $to\mathcal{T}_C$ will be inserted that, when applied to the type $\rho$, returns a chain of operations that removes, if possible, intersections in $\rho$. This can be understood by the observation that, for example,

$$((\sigma \to \sigma) \to \sigma \to \sigma) \to \sigma \text{ is a substitution instance of } ((\varphi_1 \to \varphi_1) \to \varphi_2) \cap (\varphi_3 \to \varphi_4 \to \varphi_4) \to \varphi_5.$$

Note that if quantifiers appear in $\rho$, $to\mathcal{T}_C(\rho)$ should fail, since quantifiers that appear before an arrow cannot be removed by any of the operations on types.

Finally, $unify^{\forall}{}_2\,(\sigma, S_2(\pi_2), S_2(P_2))$ is called (with $S_2 = to\mathcal{T}_C(\pi_2)$). The basis $S_2(P_2)$ is needed to calculate the expansion of $S_2(\pi_2)$ in case $\sigma$ is an intersection type.

**Definition 5.19** The function $to\mathcal{T}_C : \mathcal{T}_2 \to \mathcal{S}$ is defined by:

$$\begin{array}{lcl} to\mathcal{T}_C(\sigma) & = & [Id_S], \quad \text{if } \sigma \in \mathcal{T}_C \\ to\mathcal{T}_C((\sigma_1 \cap \cdots \cap \sigma_n) \to \mu) & = & S' \circ S_n, \text{ otherwise,} \end{array}$$

where $S_i = unify(S_{i-1}(\sigma_1), S_{i-1}(\sigma_{i+1})) \circ S_{i-1}$, $(1 \leq i \leq n-1$, with $S_0 = Id_S)$, and
$S' = to\mathcal{T}_C(S_n(\mu))$

(Again, notice that $to\mathcal{T}_C(\sigma)$ fails if $\sigma$ contains '$\forall$'.)

The algorithm $unify^{\forall}{}_2$ is called with the types $\sigma$ and $\rho'$, the latter being $\rho$ in which the intersections are removed (so $\rho' = to\mathcal{T}_C(\rho)(\rho)$; notice that $to\mathcal{T}_C(\rho)$ is an operation on types that removes all intersections in $\rho$, and needs to be applied to $\rho$). Since none of the derivation rules, nor one of the operations, allows for the removal of a quantifier that occurs *inside* a type, if $\sigma = \forall\overline{\alpha}.\sigma'$, the unification of $\sigma$ with $\rho'$ will not remove the '$\forall\overline{\alpha}$' part.

The following definition presents the main unification algorithm, $unify^{\forall}{}_2$. It gets, typically, called as

$$unify^{\forall}{}_2\,(\sigma, S_2(\pi_2), S_2(P_2))$$

during the calculation of the principal pair for an application; after deriving $P_1 \vdash^2_{\mathcal{E}} t_1 : \pi_1$ and $P_2 \vdash^2_{\mathcal{E}} t_2 : \pi_2$ as principal pairs for $t_1$ and $t_2$, respectively, with $\pi_1 = \sigma \to \tau$ and $S_2 = to\mathcal{T}_C(\pi_2)$. The basis is needed to calculate the expansion in case $\sigma$ is an intersection type, as said above.

**Definition 5.20** Let $\mathcal{B}$ be the set of all bases, and $\mathcal{C}h$ the set of all chains. The function $unify^{\forall}_2 : \mathcal{T}^{\forall}{}_C \times \mathcal{T}_C \times \mathcal{B} \to \mathcal{C}h$ is defined by:

$$\begin{array}{lcl} unify^{\forall}{}_2\,(\varphi, \tau, B) & = & (\varphi \mapsto \tau), \\ unify^{\forall}{}_2\,((\forall\overline{\alpha_1}.\sigma_1) \cap \ldots \cap (\forall\overline{\alpha_n}.\sigma_n), \tau, B) & = & [Ex, S_n], \quad \text{otherwise} \end{array}$$

$$\begin{array}{lrcl} \text{where} & Ex & = & n_{\langle B,\tau\rangle}, \\ & \tau_1 \cap \cdots \cap \tau_n & = & Ex(\tau), \text{ and} \\ & \text{for every } 1 \leq i \leq n,\ S_i & = & unify(S_{i-1}(\sigma_i), \tau_i) \circ S_{i-1} \ (\text{with } S_0 = Id_S). \end{array}$$

It is worthwhile to notice that $unify$, $to\mathcal{T}_C$, and $unify^{\forall}{}_2$ all return lifting-free chains. Moreover, both $unify$ and $to\mathcal{T}_C$ return a substitution, and the chain returned by $unify^{\forall}{}_2\,(\sigma, \tau)$ acts on $\sigma$ as a substitution: the expansion in the chain is defined for the sake of $\tau$ only. Notice also that $unify^{\forall}{}_2$ does not really return a unifying chain for its first two arguments; to achieve this, also closures would have to be inserted. They are not needed for the present purpose.

The procedure $unify^{\forall}{}_2$ fails when $unify$ fails, and $to\mathcal{T}_C$ fails when either $unify$ fails or when the argument contains '$\forall$'. Because of this relation between $unify^{\forall}{}_2$ and $to\mathcal{T}_C$ on one side, and $unify$ on the other, the procedures defined here are terminating and type assignment in the system defined in this section is decidable.

Using Property 5.18, it is possible to prove the following lemma.

*Lemma 5.21 Let Ch be a chain.*

*i) If $\sigma \in \mathcal{T}_2$, and $Ch(\sigma) = \tau \in \mathcal{T}_C$, then there is a $S'$ such that $S' \circ to\mathcal{T}_C(\sigma)(\sigma) = \tau$.*

*ii) If $\sigma \in \mathcal{T}_2$, and $Ch(\sigma) = \tau \in \mathcal{T}_1$, then there is a lifting-free chain $Ch'$ such that $[to\mathcal{T}_C(\sigma)] * Ch'(\sigma) = \tau$.*

*Proof:* Easy, using Lemma 5.7(*iii*) and (*iv*). ■

## 5.3 Principal pairs for terms

In this subsection, the principal pair for a term $t$ with respect to the environment $\mathcal{E}$ – $pp_{\mathcal{E}}(t)$ – is defined, consisting of basis $P$ and type $\pi$. In Theorem 5.29 it will be shown that, for every term, this is indeed the principal one.

Notice that, in the definition below, if $pp_{\mathcal{E}}(t) = \langle P,\pi \rangle$, then $\pi \in \mathcal{T}_2$. For example, the principal pair for the term $\lambda x.x$ is $\langle \emptyset, \varphi \rightarrow \varphi \rangle$, so, in particular, it is not $\langle \emptyset, \forall \alpha.\alpha \rightarrow \alpha \rangle$. Although one could argue that the latter type is more 'principal' in the sense that it expresses the generic character the principal type is supposed to have, we have chosen to use the former instead. This is mainly for technical reasons: because unification is used in the definition below, using the latter type, we would often be forced to remove the external quantifiers. Both types can be seen as 'principal' though, since $\forall \alpha.\alpha \rightarrow \alpha$ can be obtained from $\varphi \rightarrow \varphi$ by closure, and $\varphi \rightarrow \varphi$ from $\forall \alpha.\alpha \rightarrow \alpha$ by lifting.

**Definition 5.22** Let $t$ be a term in $T(\mathcal{F},\mathcal{X})$. Using $unify^{\forall}{}_2$, $pp_{\mathcal{E}}(t) = \langle P,\pi \rangle$, with $\pi \in \mathcal{T}_2$ is defined by:

*i)* $t \equiv x$. Then $pp_{\mathcal{E}}(x) = \langle \{x{:}\varphi\}, \varphi \rangle$.

*ii)* $t \equiv \lambda x.t'$. Let $pp_{\mathcal{E}}(t') = \langle P,\pi \rangle$, then:

*a)* If $x$ occurs free in $t'$, and $x{:}\sigma \in P$, then $pp_{\mathcal{E}}(\lambda x.t') = \langle P \backslash x, \sigma \rightarrow \pi \rangle$.

*b)* Otherwise, let $\varphi$ be a fresh variable, and $pp_{\mathcal{E}}(\lambda x.t') = \langle P, \varphi \rightarrow \pi \rangle$.

*iii)* $t \equiv Ap(t_1,t_2)$. Let $pp_{\mathcal{E}}(t_1) = \langle P_1,\pi_1 \rangle$, $pp_{\mathcal{E}}(t_2) = \langle P_2,\pi_2 \rangle$ (choose, if necessary, trivial variants such that these pairs are disjoint), and $S_2 = to\mathcal{T}_C(\pi_2)$, then either

$(\pi_1 = \varphi)$: $pp_{\mathcal{E}}(Ap(t_1,t_2)) = \langle P,\pi \rangle$, where

$$\begin{aligned} P &= S_1(\Pi\{P_1, S_2(P_2)\}), \\ \pi &= \varphi', \\ S_1 &= (\varphi \mapsto S_2(\pi_2) \rightarrow \varphi'), \text{ and} \\ \varphi' &\text{ is a fresh variable.} \end{aligned}$$

$(\pi_1 = \sigma \rightarrow \tau)$: $pp_{\mathcal{E}}(Ap(t_1,t_2)) = \langle P,\pi \rangle$, provided $P$ and $\pi$ contain no unbound occurrences of $\alpha$s, where

$$\begin{aligned} P &= S(\Pi\{P_1, Ex(S_2(P_2))\}), \\ \pi &= S(\tau), \text{ and} \\ [Ex,S] &= unify^{\forall}{}_2(\sigma, S_2(\pi_2), S_2(P_2)). \end{aligned}$$

*iv)* $t \equiv F(t_1,\ldots,t_n)$. Let, for every $1 \leq i \leq n$, $pp_{\mathcal{E}}(t_i) = \langle P_i,\pi_i \rangle$ (choose, if necessary, trivial variants such that the $\langle P_i,\pi_i \rangle$ are disjoint in pairs), then $pp_{\mathcal{E}}(F(t_1,\ldots,t_n)) = \langle P,\pi \rangle$, provided $P$ and $\pi$ contain no unbound occurrences of $\alpha$s, where

$$\begin{aligned} P &= S^n(\Pi\{Ex_1(S_1(P_1)),\ldots,Ex_n(S_n(P_n))\}, \\ \pi &= S^n(\gamma), \\ \gamma_1 \rightarrow \cdots \rightarrow \gamma_n \rightarrow \gamma &\text{ is a fresh instance of } \mathcal{E}(F), \end{aligned}$$

$$\text{and, for every } 1{\leq}i{\leq}n, \qquad \begin{array}{rcl} S_i &=& \mathit{toT}_C(\pi_i), \\ [Ex_i,S_i{}'] &=& \mathit{unify}^{\forall}{}_2(S^{i-1}(\gamma_i),S_i(\pi_i),S_i(P_i)), \text{ and} \\ S^i &=& S_i{}'{\circ}S^{i-1} \text{ (with } S^0 = Id_S). \end{array}$$

Since *unify* or $\mathit{unify}^{\forall}{}_2$ may fail, not every term has a principal pair.

Notice that closures and liftings are not needed when constructing the principal basis and type.

The treatment of $F(t_1,\ldots,t_n)$ in Definition 5.22 is in fact very much the same as a repeated treatment of $Ap(t_1,t_2)$. This can be understood by observing that, if $\mathcal{E}(F) = \sigma_1{\rightarrow}\cdots{\rightarrow}\sigma_n{\rightarrow}\sigma$, then

$$\begin{array}{lcl} pp_{\mathcal{E}}(F(x_1,\ldots,x_n)) &=& \langle\{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\},\sigma\rangle, \text{ so} \\ pp_{\mathcal{E}}(\lambda x_1\cdots x_n.F(x_1,\ldots,x_n)) &=& \langle\emptyset,\mathcal{E}(F)\rangle, \end{array}$$

and that therefore the terms

$$F(t_1,\ldots,t_n) \text{ and } Ap(\cdots Ap(Ap(\lambda x_1\cdots x_n.F(x_1,\ldots,x_n),t_1),t_2)\cdots,t_n)$$

should be treated in the same way. It is for this reason that, in the proofs below, all attention will go to the case $t = Ap(t_1,t_2)$; the case $t = F(t_1,\ldots,t_n)$ is essentially the same.

*Example 5.23* Take the rewrite rules and environment

$$\begin{array}{lcl} I(x) &\rightarrow& x \\ F(x) &\rightarrow& x \\ F(x) &\rightarrow& Ap(x,x) \end{array} \qquad \begin{array}{lcl} \mathcal{E}(I) &=& \varphi_1{\rightarrow}\varphi_1 \\ \mathcal{E}(F) &=& (\forall\alpha.\alpha{\rightarrow}\alpha){\rightarrow}\varphi_1{\rightarrow}\varphi_1 \end{array}$$

Using Definition 5.22, it is easy to check that $\vdash^2_{\mathcal{E}} F(\lambda x.I(x)){:}\,\varphi_2{\rightarrow}\varphi_2$, and $\vdash^2_{\mathcal{E}} I(\lambda x.I(x)){:}\,\varphi_2{\rightarrow}\varphi_2$. These types are the principal types for these terms.

*Example 5.24* The term $\lambda x.x(\lambda y.yy)$ does not have a principal pair. It is typeable in the full system of this paper by $(((\sigma{\rightarrow}\tau){\cap}\sigma{\rightarrow}\tau){\rightarrow}\rho){\rightarrow}\rho$ (where $B = \{x{:}((\sigma{\rightarrow}\tau){\cap}\sigma{\rightarrow}\tau){\rightarrow}\rho, y{:}(\sigma{\rightarrow}\tau){\cap}\sigma\}$).

$$\dfrac{\dfrac{\overline{B{\backslash}y \vdash_{\mathcal{E}} x{:}((\sigma{\rightarrow}\tau){\cap}\sigma{\rightarrow}\tau){\rightarrow}\rho} \qquad \dfrac{\dfrac{\overline{B \vdash_{\mathcal{E}} y{:}\sigma{\rightarrow}\tau} \qquad \overline{B \vdash_{\mathcal{E}} y{:}\sigma}}{B \vdash_{\mathcal{E}} yy{:}\tau}}{B{\backslash}y \vdash_{\mathcal{E}} \lambda y.yy{:}(\sigma{\rightarrow}\tau){\cap}\sigma{\rightarrow}\tau}}{B{\backslash}y \vdash_{\mathcal{E}} x(\lambda y.yy){:}\rho}}{\emptyset \vdash_{\mathcal{E}} \lambda x.x(\lambda y.yy){:}((\sigma{\rightarrow}\tau){\cap}\sigma{\rightarrow}\tau){\rightarrow}\rho{\rightarrow}\rho}$$

but it is not possible to type this term in $\vdash^2_{\mathcal{E}}$: since $\mathit{toT}_C((\sigma{\rightarrow}\tau){\cap}\sigma{\rightarrow}\tau)$ will fail on $\mathit{unify}(\sigma{\rightarrow}\tau,\sigma)$, $\lambda y.yy$ can only be typed with a Rank 2 type, so the type of $x$ has to be of rank 3.

*Example 5.25* The principal type for $\lambda y.yy$ is $((\varphi_1{\rightarrow}\varphi_2){\cap}\varphi_1){\rightarrow}\varphi_2$ (see Example 5.9), and the chain of operations that transforms the pair $\langle\emptyset,((\varphi_1{\rightarrow}\varphi_2){\cap}\varphi_1){\rightarrow}\varphi_2\rangle$ into the pair $\langle\emptyset,(\forall\alpha.\alpha){\rightarrow}\varphi_2\rangle$ is

$$[<\langle\emptyset,(\varphi_1{\rightarrow}\varphi_2){\cap}\varphi_1{\rightarrow}\varphi_2\rangle,\langle\emptyset,(\forall\alpha.\alpha){\rightarrow}\varphi_2\rangle>]$$

(notice that $\forall\alpha.\alpha \leq_2 (\varphi_1{\rightarrow}\varphi_2){\cap}\varphi_1$). However, this does not imply that the derivation for the latter pair is obtained by applying $(\leq)$ to the derivation for the former: $(\leq)$ can only be applied to term-variables. However, because of Lemma 5.11, we know that a derivation exists, and as can be expected, the derivation for $\emptyset \vdash_{\mathcal{E}} \lambda y.yy{:}(\forall\alpha.\alpha){\rightarrow}\varphi_2$ can be obtained by applying a lifting. First we derive the principal pair for $yy$:

$$\dfrac{\dfrac{}{\{y{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1\} \vdash_{\mathcal{E}} y : \varphi_1{\to}\varphi_2} \qquad \dfrac{}{\{y{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1\} \vdash_{\mathcal{E}} y : \varphi_1}}{\{y{:}(\varphi_1{\to}\varphi_2){\cap}\varphi_1\} \vdash_{\mathcal{E}} yy : \varphi_2}$$

to which we apply the lifting $<\langle B,\varphi_2\rangle,\langle\{y{:}\forall\alpha.\alpha\},\varphi_2\rangle>$ (notice that since this lifting changes only the basis, we can actually see it as an operation on derivations):

$$\dfrac{\dfrac{}{\{y{:}\forall\alpha.\alpha\} \vdash_{\mathcal{E}} y : \varphi_1{\to}\varphi_2} \qquad \dfrac{}{\{y{:}\forall\alpha.\alpha\} \vdash_{\mathcal{E}} y : \varphi_1}}{\{y{:}\forall\alpha.\alpha\} \vdash_{\mathcal{E}} yy : \varphi_2}$$

to which we apply $(\to I)$:

$$\dfrac{\dfrac{\dfrac{}{\{y{:}\forall\alpha.\alpha\} \vdash_{\mathcal{E}} y : \varphi_1{\to}\varphi_2} \qquad \dfrac{}{\{y{:}\forall\alpha.\alpha\} \vdash_{\mathcal{E}} y : \varphi_1}}{\{y{:}\forall\alpha.\alpha\} \vdash_{\mathcal{E}} yy : \varphi_2}}{\emptyset \vdash_{\mathcal{E}} \lambda y.yy : (\forall\alpha.\alpha){\to}\varphi_2}$$

The following lemmas are needed in the proof of Theorem 5.29. The first states that if a chain maps the principal pairs of terms $t_1,t_2$ in an application $Ap(t_1,t_2)$ to pairs that allow the application itself to be typed, then these pairs can also be obtained by first performing a unification. The second generalises this result to arbitrary function applications.

*Lemma 5.26 Let $\sigma \in \mathcal{T}_2$, and for $i = 1,2$, $pp_{\mathcal{E}}(t_i) = \langle P_i,\pi_i\rangle$, such that these pairs are disjoint. Let $Ch_1,Ch_2$ be chains such that*

$$Ch_1(pp_{\mathcal{E}}(t_1)) = \langle B_1,\sigma{\to}\tau\rangle, \text{ and } Ch_2(pp_{\mathcal{E}}(t_2)) = \langle B_2,\sigma\rangle.$$

*Then there are a chains $Ch_u$ and $Ch_p$, and type $\rho \in \mathcal{T}_2$ such that*

$$\begin{aligned} pp_{\mathcal{E}}(Ap(t_1,t_2)) &= Ch_u(\langle \Pi\{P_1,P_2\},\rho\rangle), \text{ and} \\ Ch_p(pp_{\mathcal{E}}(Ap(t_1,t_2))) &= \langle \Pi\{B_1,B_2\},\tau\rangle. \end{aligned}$$

*Proof:* Since $Ch_2(\pi_2) = \sigma \in \mathcal{T}_1$, by Lemma 5.21(*ii*), $Ch_2 = Ch_2 * [to\mathcal{T}_C(\pi_2)]$, so $S_2 = to\mathcal{T}_C(\pi_2)$ exists; let $\pi'_2 = S_2(\pi_2)$, and $P'_2 = S_2(P_2)$. We distinguish the cases:

$(\pi_1 = \varphi)$: Then $pp_{\mathcal{E}}(Ap(t_1,t_2)) = \langle S_1(\Pi\{P_1,P'_2\}),\varphi'\rangle$, where $S_1 = (\varphi \mapsto \pi'_2{\to}\varphi')$, and $\varphi'$ is a fresh variable; take $Ch_u = [S_1,S_2]$.

$$Ch_1(\langle P_1,\varphi\rangle) = \langle B_1,\sigma{\to}\tau\rangle, \text{ and } Ch_2(\langle P_2,\pi_2\rangle) = \langle B_2,\sigma\rangle.$$

Since $\pi_2 \in \mathcal{T}_2$ and $Ch_2(\pi_2) \in \mathcal{T}_1$, by Lemma 5.21(*ii*) and Lemma 5.7(*ii*), there exists a lifting-free chain $Ch'_2$ such that $\sigma = Ch'_2 * [S_2](\pi_2) = Ch'_2(\pi'_2)$.

Also, $Ch_1(\varphi) = \sigma{\to}\tau$, so there exists a chain $Ch'_1$ such that $Ch_1 = Ch'_1 * [(\varphi \mapsto \varphi_1{\to}\varphi_2)]$, so such that $Ch'_1(\varphi_1) = \sigma$, and $Ch'_1(\varphi_2) = \tau$.

Since $Ch_1(\pi_1) = \sigma = Ch'_2 * [S_1,S_2](\pi_2)$, in particular

To show: there exists $Ch_p$ such that $Ch_p(pp_{\mathcal{E}}(Ap(t_1,t_2))) = \langle \Pi\{B_1,B_2\},\tau\rangle$, so such that $Ch_p(\langle S_1(\Pi\{P_1,P'_2\}),\varphi'\rangle) = \langle \Pi\{B_1,B_2\},\tau\rangle$.

$(\pi_1 = \mu_1{\to}\mu_2)$: Notice that $Ch_1(\pi_1) = Ch_1(\mu_1{\to}\mu_2) = \sigma{\to}\tau \in \mathcal{T}_2$, so, by Lemma 5.7(*v*), there are substitution $S$, and lifting $L$ such that

$$Ch_1(\pi_1) = [S,L](\pi_1).$$

So there are $B' \geq B$, $\sigma \leq \rho_1$, $\rho_2 \leq \tau$ such that

$$\begin{array}{rcl} S(\pi_1) = S(\mu_1 \rightarrow \mu_2) & = & \rho_1 \rightarrow \rho_2, \text{ (So } S(\mu_1) = \rho_1\text{), and} \\ L & = & <\langle B', \rho_1 \rightarrow \rho_2 \rangle, \langle B, \sigma \rightarrow \tau \rangle>. \end{array}$$

In particular,

$$\begin{array}{rcll} \pi_1 & = & \mu_1 \rightarrow \mu_2, \text{ so} & (\mu_1 \in \mathcal{T}_1, \mu_2 \in \mathcal{T}_2) \\ \pi_1 & = & (\forall \overline{\alpha_1}.\delta_1 \cap \cdots \cap \forall \overline{\alpha_m}.\delta_m) \rightarrow \mu_2, \text{ so} & (\delta_j \in \mathcal{T}_C \ (1 \leq j \leq m),\ \mu_2 \in \mathcal{T}_2) \\ S(\pi_1) & = & (\forall \overline{\alpha_1}.S(\delta_1) \cap \cdots \cap \forall \overline{\alpha_m}.S(\delta_m)) \rightarrow S(\mu_2) \ = \ \rho_1 \rightarrow \rho_2. & \end{array}$$

Notice that $<\langle B, \sigma \rangle, \langle B, \rho_1 \rangle>$ is a lifting, and $Ch_2 * [<\langle B, \sigma \rangle, \langle B, \rho_1 \rangle>](\pi_2) = \rho_1 \in \mathcal{T}_1$. Then, by Lemma 5.21(*ii*), there is a lifting-free chain $Ch'$ such that $Ch'(\pi'_2) = \rho_1$. Then

$$\begin{array}{rcllll} Ch' & = & [Ex', S', \overline{Cl}], & \text{with } Ex'(\pi'_2) & = & S'_1(\pi'_2) \cap \cdots \cap S'_n(\pi'_2), \quad (\forall \text{ free}) \\ [Ex', S'](\langle P'_2, \pi'_2 \rangle) & = & \langle B_1, \nu \rangle, & \text{with} \quad \nu & = & S'(S'_1(\pi'_2)) \cap \cdots \cap S'(S'_n(\pi'_2)), \quad '' \\ \overline{Cl}(\langle B, \nu \rangle) & = & \langle B, \rho_1 \rangle, & \text{with} \quad \rho_1 & = & \forall \overline{\alpha_1}.\nu_2^1 \cap \cdots \cap \forall \overline{\alpha_m}.\nu_2^m \end{array}$$

So, for $1 \leq j \leq m$, $S(\delta_j) = \nu_2^j = S'(S'_j(\pi'_2))$, and, by Property 5.18, there exists substitutions $S_u^j, S^j$ such that

$$\begin{array}{c} S_u^j \ = \ \mathit{unify}(\delta_j, \pi'_2) \\ S(\delta_j) = S^j(S_u^j(\delta_j)) \ = \ \nu_2^j \ = S^j(S_u^j(\pi'_2)) = S'(S'_j(\pi'_2)) \end{array}$$

Since the substitutions $S_u^j$ agree on type-variables, without loss of generality, we can assume that exists substitutions $S_1, \ldots, S_m$ such that

$$\begin{array}{rcl} S_i & = & \mathit{unify}(S_{i-1}(\delta_i), \pi'_2) \circ S_{i-1} \text{ (with } S_0 = Id_S\text{), and} \\ \text{for } 1 \leq j \leq m,\ S^j(S_n(\delta_j)) & = & \nu_2^j = S^j(S_n(\pi'_2)) \end{array}$$

so, by Definition 5.20

$$Ch'_u = \mathit{unify}^{\forall}{}_2((\forall \overline{\alpha_1}.\delta_1) \cap \ldots \cap (\forall \overline{\alpha_n}.\delta_n), \pi'_2, P'_2).$$

exists. Take $Ch_u = [S_2] * Ch'_u$, $Ch_p = [S^1 \circ \cdots \circ S^m, <\langle B, \rho_2 \rangle, \langle B, \tau \rangle>]$, and $\rho = \mu_2$. ■

Notice that, since $B$ can be assumed to not contain free occurrences of $\alpha$s, the last chain is well-defined.

*Lemma 5.27 Let $\sigma \in \mathcal{T}_2$, and, for every $1 \leq i \leq n$, $pp_{\mathcal{E}}(t_i) = \langle P_i, \pi_i \rangle$, such that the pairs $\langle P_i, \pi_i \rangle$ and the type $\mathcal{E}(F) = \gamma_1 \rightarrow \cdots \rightarrow \gamma_n \rightarrow \gamma$ are disjoint, and let $Ch(F), Ch_1, \ldots, Ch_n$ be chains such that*

$$Ch_F(\mathcal{E}(F)) = \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \sigma \text{ and, for every } 1 \leq i \leq n, Ch_i(\langle P_i, \pi_i \rangle) = \langle B_i, \sigma_i \rangle.$$

*Then there are chains $Ch_g$ and $Ch_p$ such that*

$$\begin{array}{rcl} pp_{\mathcal{E}}(F(t_1, \ldots, t_n)) & = & Ch_g(\langle \Pi\{P_1, \ldots, P_n\}, \gamma \rangle), \text{ and} \\ Ch_p(pp_{\mathcal{E}}(F(t_1, \ldots, t_n))) & = & \langle \Pi\{B_1, \ldots, B_n\}, \sigma \rangle. \end{array}$$

*Proof:* This is a generalisation of the proof of the previous lemma. ■

The main result of this section then becomes the soundness and completeness result for $pp_{\mathcal{E}}$.

**Theorem 5.28** (Soundness of $pp_{\mathcal{E}}$) *If $pp_{\mathcal{E}}(t) = \langle P, \pi \rangle$, then $P \vdash^2_{\mathcal{E}} t : \pi$.*

*Proof:* By induction on the structure of terms.

$(t \equiv x)$: Since $pp_{\mathcal{E}}(x) = \langle \{x{:}\varphi\}, \varphi \rangle$, the result follows from rule $(\leq_2)$.

$(t \equiv \lambda x.t')$: *a*) If $x$ occurs free in $t'$, then there are $\sigma, \pi$ such that $pp_{\mathcal{E}}(\lambda x.t') = \langle P, \sigma \rightarrow \pi \rangle$, where $pp_{\mathcal{E}}(t') = \langle P, x{:}\sigma, \pi \rangle$. By induction, $P, x{:}\sigma \vdash^2_{\mathcal{E}} t' : \pi$, and, by applying rule $(\rightarrow I)$, also $P \vdash^2_{\mathcal{E}} \lambda x.t' : \sigma \rightarrow \pi$.

*b*) Otherwise, there are $\varphi, \pi$ such that $pp_{\mathcal{E}}(\lambda x.t') = \langle P, \varphi \rightarrow \pi \rangle$, where $pp_{\mathcal{E}}(t') = \langle P, \pi \rangle$, with $\varphi$ a fresh variable. By induction, $P \vdash^2_{\mathcal{E}} t' : \pi$. Since $x$ does not occur in $P$, we have $P, x{:}\varphi \leq_2 P$, and $P, x{:}\varphi \vdash^2_{\mathcal{E}} t' : \pi$ by Lemma 5.11, and then, by rule $(\rightarrow I)$, $P \vdash^2_{\mathcal{E}} \lambda x.t' : \varphi \rightarrow \pi$.

$(t \equiv Ap(t_1, t_2))$: Then there are $P_1, P_2$ such that $pp_{\mathcal{E}}(t_1) = \langle P_1, \pi_1 \rangle$, $pp_{\mathcal{E}}(t_2) = \langle P_2, \pi_2 \rangle$, and, by induction, $P_1 \vdash^2_{\mathcal{E}} t_1 : \pi_1$ and $P_2 \vdash^2_{\mathcal{E}} t_2 : \pi_2$. Notice that, by construction, the two principal pairs share no type variables. Let $S_2 = to\mathcal{T}_C(\pi_2)$. Then either:

$(\pi_1 = \varphi)$: Then $pp_{\mathcal{E}}(Ap(t_1, t_2)) = \langle S_1(\Pi\{P_1, S_2(P_2)\}), \varphi' \rangle$, where $S_1 = (\varphi \mapsto S_2(\pi_2) \rightarrow \varphi')$, and $\varphi'$ is a fresh variable. Let $S = S_1 \circ S_2$, then, by Lemma 5.10,

$$S(P_1) \vdash^2_{\mathcal{E}} t_1 : S(\pi_1) \text{ and } S(P_2) \vdash^2_{\mathcal{E}} t_2 : S(\pi_2).$$

We know that $S(\pi_1) = S_1(\varphi) = S_2(\pi_2) \rightarrow \varphi'$. Since $S_2$ has no effect on $P_1$, $S(\Pi\{P_1, P_2\}) = S_1(\Pi\{P_1, S_2(P_2)\})$. Also, since $S_1$ does not affect any variable in $S_2(\langle P_2, \pi_2 \rangle)$, we have $S(P_2) = S_2(P_2)$, and $S(\pi_2) = S_2(\pi_2)$. So, by rule $(\rightarrow E)$, $S(\Pi\{P_1, P_2\}) \vdash^2_{\mathcal{E}} Ap(t_1, t_2) : \varphi'$.

$(\pi_1 = \sigma \rightarrow \tau$, *with* $\sigma \in \mathcal{T}_1$, $\tau \in \mathcal{T}_2)$: Then $pp_{\mathcal{E}}(Ap(t_1, t_2)) = \langle S(\Pi\{P_1, Ex(S_2(P_2))\}), S(\tau) \rangle$, where $[Ex, S] = unify^{\forall}{}_2(\sigma, S_2(\pi_2), S_2(P_2))$. Let $Ch = [S_2, Ex, S]$, then, by Lemma 5.15,

$$Ch(P_1) \vdash^2_{\mathcal{E}} t_1 : Ch(\pi_1) \text{ and } Ch(P_2) \vdash^2_{\mathcal{E}} t_2 : Ch(\pi_2).$$

Since $[Ex, S](\sigma) = [Ex, S](S_2(\pi_2))$, and $S_2$ has no effect on $\langle P_1, \sigma \rightarrow \tau \rangle$, we have $Ch(\sigma) = Ch(\pi_2)$ and $S(\Pi\{P_1, Ex(S_2(P_2))\}) = Ch(\Pi\{P_1, P_2\})$. Also, since the chain returned by $unify^{\forall}{}_2$ acts on $\langle P_1, \sigma \rightarrow \tau \rangle$ as a substitution, we get $Ch(\sigma \rightarrow \tau) = Ch(\pi_2) \rightarrow S(\tau)$. But then, by rule $(\rightarrow E)$, $Ch(\Pi\{P_1, P_2\}) \vdash^2_{\mathcal{E}} Ap(t_1, t_2) : S(\tau)$.

$(t \equiv F(t_1, \ldots, t_n))$: As the previous part, using that the fresh instance of the environment type for $F$ is can be used for $F$ here (see the proof of Lemma 5.10). ∎

**Theorem 5.29** (Completeness of $pp_{\mathcal{E}}$) *If $B \vdash^2_{\mathcal{E}} t : \sigma$, then there are a basis $P$ and type $\pi$ such that $pp_{\mathcal{E}}(t) = \langle P, \pi \rangle$, and there is a chain $Ch$ such that $Ch(\langle P, \pi \rangle) = \langle B, \sigma \rangle$.*

*Proof:* By induction on the structure of derivations.

$(\leq_2)$: Then $t \equiv x$, $\sigma \in \mathcal{T}_C$, and there is $\tau \in \mathcal{T}_1$ such that $x{:}\tau \in B$ and $\tau \leq_2 \sigma$. Also, $\pi = \varphi$ and $P = \{x{:}\varphi\}$. Since $\tau \leq_2 \sigma$, $B \leq_2 \{x{:}\sigma\}$, so $<\langle \{x{:}\sigma\}, \sigma \rangle, \langle B, \sigma \rangle>$ is a lifting. Take $Ch = [\varphi \mapsto \sigma, <\langle \{x{:}\sigma\}, \sigma \rangle, \langle B, \sigma \rangle>]$.

$(\cap I)$: Then $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, and, for $1 \leq i \leq n$, $B \vdash^2_{\mathcal{E}} t : \sigma_i$, with $\sigma_i \in \mathcal{T}^{\forall}{}_C$. Let $pp_{\mathcal{E}}(t) = \langle P, \pi \rangle$, and let $Ex = n_{\langle P, \pi \rangle}$, then $Ex(\langle P, \pi \rangle) = \langle \Pi\{P_1, \ldots, P_n\}, \pi_1 \cap \cdots \cap \pi_n \rangle$, with each pair $\langle P_i, \pi_i \rangle$ a trivial variant of $\langle P, \pi \rangle$. So, without loss of generality, we can even say $pp_{\mathcal{E}}(t) = \langle P_i, \pi_i \rangle$, for all $1 \leq i \leq n$. By induction, there exist chains $Ch_1, \ldots, Ch_n$ such that for $1 \leq i \leq n$, $Ch_i(\langle P_i, \pi_i \rangle) = \langle B, \sigma_i \rangle$. By Lemma 5.7(*ii*), $Ch_i = [S_i, \overline{Cl_i}]$. Take $Ch = [Ex, S_1 \circ \cdots \circ S_n, \overline{Cl_n}, \ldots, \overline{Cl_n}]$.

$(\rightarrow I)$: Then $t \equiv \lambda x.t'$, and there are $\rho, \mu$ such that $\sigma = \rho \rightarrow \mu$, and $B, x{:}\rho \vdash^2_{\mathcal{E}} t' : \mu$. Let $pp_{\mathcal{E}}(t') = \langle P', \pi' \rangle$, then

$(x \in FV(t'))$: Let $x{:}\nu \in P'$, then $P = P' \backslash x$, and $pp_{\mathcal{E}}(\lambda x.t') = \langle P, \nu \rightarrow \pi' \rangle$. By induction there exists a chain $Ch'$ such that

$$Ch'(\langle P, x{:}\nu, \pi' \rangle) = \langle B, x{:}\rho, \mu \rangle.$$

Then, by Lemma 5.16($i$), there exists a chain $Ch''$ such that $Ch''(\langle P,\nu\rightarrow\pi'\rangle) = \langle B,\rho\rightarrow\mu\rangle$. Take $Ch = Ch''$.

$(x \notin FV(t'))$: Then $pp_{\mathcal{E}}(\lambda x.t') = \langle P',\varphi\rightarrow\pi'\rangle$, where $\varphi$ is a type-variable not occurring in any other type. By induction there exists a chain $Ch'$ such that $Ch'(\langle P,\pi\rangle) = \langle B,\mu\rangle$. Then, by Lemma 5.16($ii$), there exists a chain $Ch''$ such that $Ch''(\langle P,\varphi\rightarrow\pi\rangle) = \langle B,\varphi\rightarrow\mu\rangle$. Let $Ch'' = [Ex,S,L,\overline{Cl}]$, then take $Ch = [Ex,S\circ(\varphi\mapsto\rho),L,\overline{Cl}]$.

$(\rightarrow E)$: Then $t \equiv Ap(t_1,t_2)$, and there is a $\tau \in \mathcal{T}_2$ such that $B \vdash^2_{\mathcal{E}} t_1{:}\tau\rightarrow\sigma$, and $B \vdash^2_{\mathcal{E}} t_2{:}\tau$. By induction, for $i = 1$, 2, there are $P_i,\pi_i$, and chain $Ch_i$ such that

$$pp_{\mathcal{E}}(t_i) = \langle P_i,\pi_i\rangle,\ Ch_1(pp_{\mathcal{E}}(t_1)) = \langle B,\tau\rightarrow\sigma\rangle, \text{ and } Ch_2(pp_{\mathcal{E}}(t_2)) = \langle B,\tau\rangle.$$

Then, by Lemma 5.26, there is a chain $Ch$ such that $Ch(pp_{\mathcal{E}}(Ap(t_1,t_2))) = \langle B,\sigma\rangle$.

$(\forall I)$: Then $\sigma = \forall\alpha.\tau[\alpha/\varphi]$, and $B \vdash^2_{\mathcal{E}} t{:}\tau$. Let $pp_{\mathcal{E}}(t) = \langle P,\pi\rangle$, then by induction there exists a chain $Ch'$ such that $Ch'(\langle P,\pi\rangle) = \langle B,\tau\rangle$. Take $Cl = \langle\tau,\varphi\rangle$, and $Ch = Ch' * [Cl]$.

$(\mathcal{F})$: Then $t \equiv F(t_1,\ldots,t_n)$. There are $\sigma_1,\ldots,\sigma_n$ such that, for every $1 \leq i \leq n$, $B \vdash^2_{\mathcal{E}} t_i{:}\sigma_i$, and a chain $Ch_F$ such that $Ch_F(\mathcal{E}(F)) = \sigma_1\rightarrow\cdots\rightarrow\sigma_n\rightarrow\sigma$. By induction, for $1 \leq i \leq n$, there are $\langle P_i,\pi_i\rangle$, (disjoint in pairs) and a chain $Ch_i$, such that

$$pp_{\mathcal{E}}(t_i) = \langle P_i,\pi_i\rangle, \text{ and } Ch_i(pp_{\mathcal{E}}(t_i)) = \langle B,\sigma_i\rangle.$$

Since the pairs $\langle P_i,\pi_i\rangle$ are disjoint, the chains $Ch_i$ do not interfere. Assume, without loss of generality, that none of the type-variables occurring in $\mathcal{E}(F)$ occur in any of the pairs $\langle P_i,\pi_i\rangle$. Then, by Lemma 5.27, there is a chain $Ch$ such that $Ch(pp_{\mathcal{E}}(F(t_1,\ldots,t_n))) = \langle B,\sigma\rangle$. ■

## 6 Conclusions

In this paper we made a further step inside the partially unexplored field of type assignment systems for term rewriting systems extended with abstraction and $\beta$-rule. Various extensions of term rewriting containing the notions of abstraction and $\beta$-rule have been intensively explored in the last decade in a plethora of papers leaving, however, the concept of type assignment in such a context still partially in darkness. We believe that if the study of the combined computational paradigm Lambda Calculus and Algebraic Rewriting can be of help not only for the development of interactive proof assistant tools, but also as basis for powerful and expressive programming languages, the topic of type systems, and type assignment systems in particular, cannot be left unattended. We have focused on a type assignment aiming a being both powerful, by using intersection types, and expressive, by adding the universal type quantification.

We cannot help to sympathize with the reader who had to go through the heavy technicalities that the notion of intersection types brings, particularly when connected with term rewriting and universal quantification. But this has not to leave the impression of a uselessness of the sort of type assignment we propose. The typing power of our general system is such to leave space to many kind of restrictions, needed also because of its undecidability. As a matter of fact, we proposed a decidable restriction which, besides having such feature, gets rid of many technicalities of the general system. We do not claim such restriction to be "the restriction" for a practical use of the proposed notion of type assignment for Lambda Calculus + Algebraic Rewriting, but simply wished to show one possibility.

The present paper then aims at being the theoretical platform from which we can start for further investigations, which will be focused on restricted versions of the general system. One other question to be addressed, directly in the context of some restricted version, is if the proposed notion of type assignment easily extends to other type constructors often used in actual type system for programming languages (e.g. the polymorphic lists or the restriction to type classes of the language Haskell).

**Acknowledgements**

The second author would like to thank Mariangiola Dezani and Carmelo Dell'Acqua for all they taught him in different fields.

## References

[1] Z. Ariola, R. Kennaway, J.W. Klop, R. Sleep, and F-J. de Vries. Syntactic definitions of undefined: on defining the undefined. In M. Hagiya and J.C. Mitchell, editors, *Proceedings of TACS '94. International Symposium on Theoretical Aspects of Computer Software,* Sendai, Japan, volume 789 of *Lecture Notes in Computer Science*, pages 543–554. Springer-Verlag, 1994.

[2] S. van Bakel. Complete restrictions of the Intersection Type Discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.

[3] S. van Bakel. Partial Intersection Type Assignment in Applicative Term Rewriting Systems. In M. Bezem and J.F. Groote, editors, *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications,* Utrecht, the Netherlands, volume 664 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag, 1993.

[4] S. van Bakel. Principal type schemes for the Strict Type Assignment System. *Logic and Computation*, 3(6):643–670, 1993.

[5] S. van Bakel. Intersection Type Assignment Systems. *Theoretical Computer Science*, 151(2):385–435, 1995.

[6] S. van Bakel. Rank 2 Intersection Type Assignment in Term Rewriting Systems. *Fundamenta Informaticae*, 2(26):141–166, 1996.

[7] S. van Bakel, F. Barbanera, and M. Fernández. Rewrite Systems with Abstraction and $\beta$-rule: Types, Approximants and Normalization. In Hanne Riis Nielson, editor, *Programming Languages and Systems – ESOP'96. Proceedings of 6th European Symposium on Programming,* Linköping, Sweden, volume 1058 of *Lecture Notes in Computer Science*, pages 387–403. Springer-Verlag, 1996.

[8] S. van Bakel and M. Fernández. Normalization Results for Typeable Rewrite Systems. *Information and Computation*, 133(2):73–116, 1997.

[9] F. Barbanera and M. Fernández. Intersection Type Assignment Systems with Higher-Order Algebraic Rewriting. *TCS*, 170:173–207, 1996.

[10] F. Barbanera, M. Fernández, and H. Geuvers. Modularity of Strong Normalization and Confluence in the Algebraic $\lambda$-cube. In *Proceedings of the ninth Annual IEEE Symposium on Logic in Computer Science,* Paris, France, 1994.

[11] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

[12] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[13] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science*, pages 82–90, 1988.

[14] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1):3–28, 1991.

[15] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. *Information and Computation*, 82:3–28, 1992.

[16] A. Bucciarelli, S. De Lorenzis, A. Piperno, and I. Salvo. Some computational properties of intersection types (extended abstract). In *Proc. Symposium on Logic in Computer Science (LICS'99)*, pages 109–118, 1999.

[17] L.M.M. Damas. *Type Assignment in Programming Languages*. PhD thesis, University of Edinburgh, Department of Computer Science, Edinburgh, 1985. Thesis CST-33-85.

[18] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 245–320. North-Holland, 1990.

[19] D. J. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101:251–267, 1992.

[20] M. Fernández. Type Assignment and Termination of Interaction Nets. *Mathematical Structures of Computer Science*, 1998.

[21] M. Fernández and J.P. Jouannaud. Modular termination of term rewriting systems revisited. In E. Astesiano and A. Tarlecki, editors, *Recent Trends in Data Type Specification. 10th Workshop on Specification of Abstract Data Types,* S. Margherita, Italy, volume 906 of *Lecture Notes in Computer Science*, pages 255–272. Springer-Verlag, 1994.

[22] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[23] J.Y. Girard. The System F of Variable Types, Fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[24] B. Jacobs, I. Margaria, and M. Zacchi. Filter models with polymorphic types. *Theoretical Computer Science*, 95:143–158, 1992.

[25] T. Jim. What are principal typings and what are they good for? In *Proceedings 23th ACM Symposium on Principles of Programming Languages*, 1996.

[26] J.P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 350–361, 1991.

[27] A.J. Kfoury and J. Tiuryn. Type reconstruction in finite-rank fragments of the second-order $\lambda$-calculus. *Information and Computation*, 98(2):228–257, 1992.

[28] A.J. Kfoury and J. Wells. A Direct Algorithm for Type Inference in the Rank-2 Fragment of the Second-Order $\lambda$-Calculus. In *LFP'94*, 1994.

[29] A.J. Kfoury and J.B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Proceedings 26th ACM Symposium on Principles of Programming Languages*, pages 161–174, 1999.

[30] J.W. Klop. Term Rewriting Systems. In S. Abramsky, Dov.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–116. Clarendon Press, 1992.

[31] J. Launchbury and S.L. Peyton Jones. Lazy functional state threads. In *Proc. ACM SIGPLAN'94 Conference on Programming Language Design and Implementation*, 1994.

[32] I. Margaria and M. Zacchi. Principal Typing in a $\forall\cap$-Discipline. *Logic and Computation*, 5(3):367–381, 1995.

[33] M.H.A. Newman. On theories with a combinatorial definition of 'equivalence'. *Ann. Math.*, 43:223–243, 1942.

[34] E.G.J.M.H. Nöcker, J.E.W. Smetsers, M.C.J.D. van Eekelen, and M.J. Plasmeijer. Concurrent Clean. In *Proceedings of PARLE '91, Parallel Architectures and Languages Europe,* Eindhoven, The Netherlands, volume 506-II of *Lecture Notes in Computer Science*, pages 202–219. Springer-Verlag, 1991.

[35] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[36] S. Ronchi Della Rocca and B. Venneri. Principal type schemes for an extended type theory. *Theoretical Computer Science*, 28:151–169, 1984.

[37] H. Yokohuchi. Embedding a Second-Order Type System into an Intersection Type System. *Information and Computation*, 117:206–220, 1995.

## Appendix

*Property* 3.9 Let $B \vdash_{\mathcal{E}} t{:}\sigma$, where $B = \{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\}$, and let $FV(\sigma_1,\ldots,\sigma_n,\sigma) \subseteq \{\overline{\varphi}\}$. Then, given a sequence of types $\overline{\gamma}$ such that $|\overline{\gamma}| = |\overline{\varphi}|$ and a sequence of reducibility candidates $\overline{\mathcal{R}^{\gamma}}$, if R is a term-substitution that is $\overline{\mathcal{R}^{\gamma}}$-reducible for $B$, then $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

*Proof:* By Nötherian induction on '$\gg^{SN}$'. If $\sigma \equiv \sigma_1\cap\cdots\cap\sigma_n$, then, by definition, we have to prove that, for any $1\leq i\leq n$, $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma_i}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Thus, without loss of generality we can consider $\sigma$ not to be an intersection.

Let $\overline{\gamma}$ and $\overline{\mathcal{R}^{\gamma}}$ be as in the hypothesis of the property and let $\mathrm{R} = \{x_1 \mapsto u_1,\ldots,x_n \mapsto u_n\}$. We distinguish the cases:

*i)* $t$ is a neutral term. If $t$ is a variable $x_j$, then we have necessarily that $\sigma_j \leq \sigma$. Since R is $\overline{\mathcal{R}^{\gamma}}$-reducible for $B$, ${x_j}^{\mathrm{R}} \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, and, by Lemma 3.6, we have also that ${x_j}^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. So, without loss of generality we can assume that $t$ is not a variable. This implies that also $t^{\mathrm{R}}$ is neutral. If $t^{\mathrm{R}}$ is irreducible, then $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ holds by (*C3*). Otherwise, let $t^{\mathrm{R}} \to t'$ at position $p$. We will prove either $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ itself, or prove $t' \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ and apply (*C3*).

*a)* $p = qp'$, where $t|_q \equiv x_i \in \mathcal{X}$. So the redex is in a sub-term of $t^{\mathrm{R}}$ that is introduced by the term-substitution. Let $z$ be a new variable.

Take now R$'$ such that $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto t'|_q\}$. Note that $t^{\mathrm{R}}|_q \to t'|_q$ at position $p'$. Since $t^{\mathrm{R}}|_q \in \{\mathrm{R}\}$ and R is assumed to be $\overline{\mathcal{R}^{\gamma}}$-reducible for $B$, also $t^{\mathrm{R}}|_q \in \mathsf{Red}^{\sigma_i}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ holds. So, by (*C2*) we have that also $t'|_q \in \mathsf{Red}^{\sigma_i}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ holds. Then we have that R$'$ is $\overline{\mathcal{R}^{\gamma}}$-reducible for $B,z{:}\sigma_i$.

Now, if the variable $x_i$ ($\equiv t|_q$) has exactly *one* occurrence in $t$, then $t \equiv t[z]_q$ modulo renaming of variables. Otherwise, $t \trianglerighteq t[z]_q$. In the first case (since R contains a term that is rewritten to get R$'$) we have $\mathcal{I}(t^{\mathrm{R}}) \gg_3^{SN} \mathcal{I}(t[z]_q^{\mathrm{R}'})$, and $\mathcal{I}(t^{\mathrm{R}}) \gg_2^{SN} \mathcal{I}(t[z]_q^{\mathrm{R}'})$ in the second case. Both cases yield, by induction, $t[z]_q^{\mathrm{R}'} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Note that $t[z]_q^{\mathrm{R}'} \equiv t'$.

*b)* Now assume that $p$ is a non-variable position in $t$. We analyze separately the cases:

*1)* $p$ is not the root position. Note that $t|_p{}^{\mathrm{R}} \equiv t^{\mathrm{R}}|_p$. Let $\tau_k$ ($k \in K$) be a type assigned to $t|_p$ in the derivation of $B \vdash_{\mathcal{E}} t{:}\sigma$, then

$$t^{\mathrm{R}}|_p \in \mathsf{Red}^{\tau_k}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \tag{1}$$

holds by induction. Let $z$ be a new variable, and take R$'$ such that $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto t^{\mathrm{R}}|_q\}$. By (1), and since

$$\mathsf{Red}^{\bigcap_{k\,\in\,K}\tau_k}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}] \equiv \bigcap_{k\,\in\,K} \mathsf{Red}^{\tau_k}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}],$$

R$'$ is reducible for $B \cup \{z{:}\bigcap_{k\,\in\,K}\tau_k\}$. Moreover $B \cup \{z{:}\bigcap_{k\,\in\,K}\tau_k\} \vdash_{\mathcal{E}} t[z]_p{:}\sigma$. Now, since, $t \trianglerighteq t[z]_p$, we have that $\mathcal{I}(t^{\mathrm{R}}) \gg_2^{SN} \mathcal{I}(t[z]_p^{\mathrm{R}'})$, and hence $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ because $t^{\mathrm{R}} \equiv t[z]_p^{\mathrm{R}'}$ and, by induction, $t[z]_p^{\mathrm{R}'} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

*2)* $p$ is the root position. Then the possible cases for $t$ are:

*A)* $t \equiv F(t_1\ldots t_n)$, where at least one of the $t_i$'s is not a variable, and $F$ is either a defined symbol of arity $n$, or $F \equiv Ap$ and $n = 2$. Take now $z_1,\ldots,z_n$ new variables and R$'$ such that $\mathrm{R}' = \mathrm{R} \cup \{z_1 \mapsto {t_1}^{\mathrm{R}},\ldots,z_n \mapsto {t_n}^{\mathrm{R}}\}$. Since $t \trianglerighteq t_i$, $\mathcal{I}(t^{\mathrm{R}}) \gg_2^{SN} \mathcal{I}({t_i}^{\mathrm{R}})$. Then if $B \vdash_{\mathcal{E}} t_i{:}\xi_i$, ${t_i}^{\mathrm{R}} \in \mathsf{Red}^{\xi_i}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ holds by induction, and hence R$'$ is $\overline{\mathcal{R}^\gamma}$-reducible for $B \cup \{z_1{:}\xi_1,\ldots,z_n{:}\xi_n\}$. Since $t \rhd F(z_1,\ldots,z_n)$, we have $\mathcal{I}(t^{\mathrm{R}}) \gg_2^{SN} \mathcal{I}(F(z_1,\ldots,z_n)^{\mathrm{R}'})$. Now, $F(z_1,\ldots,z_n)^{\mathrm{R}'} \equiv t^{\mathrm{R}}$ and

$$B \cup \{z_1{:}\xi_1,\ldots,z_n{:}\xi_n\} \vdash_{\mathcal{E}} F(z_1,\ldots,z_n){:}\sigma.$$

Hence, by induction, $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$.

*B*) $t \equiv F^k(z_1,\ldots,z_n)$ where $z_1,\ldots,z_n$ are different variables. (If $z_i \equiv z_j$ for some $i \neq j$, we can reason as in the case above, taking $z'_1,\ldots,z'_n$, new pair-wise distinct variables, and $\mathrm{R}' = \{z'_1 \mapsto z_1{}^{\mathrm{R}},\ldots,z'_n \mapsto z_n{}^{\mathrm{R}}\}$.) Then $t^{\mathrm{R}}$ must be an instance of the left-hand side of a rule defining $F^k$, that is a rule of the form

$$F^k(\overline{\mathrm{C}[\overline{x}]},\overline{y}) \to \mathrm{C}'[F^k(\overline{\mathrm{C}1\overline{x}]},\overline{y}),\ldots,F^k(\overline{\mathrm{C}_m[\overline{x}]},\overline{y}),\overline{y}].$$

Therefore we have

$$\begin{aligned} t^{\mathrm{R}} &\equiv F^k(z_1,\ldots,z_n)^{\mathrm{R}} \\ &\equiv F^k(\overline{\mathrm{C}[\overline{v}]},\overline{w}) \\ &\to \mathrm{C}'[F^k(\overline{\mathrm{C}1\overline{v}},\overline{w}),\ldots,F^k(\overline{\mathrm{C}_m[\overline{v}]},\overline{w}),\overline{w}] \\ &\equiv t', \end{aligned}$$

where $\overline{\mathrm{C}[\overline{v}]},\overline{w}$ are all terms in $\{\mathrm{R}\}$, and hence, by hypothesis, for suitable $i$'s, they belong to $\mathsf{Red}^{\sigma_i}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$. Now we will deduce $t' \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$ in three steps:

(*Step I*): Let R$'$ be the term-substitution that maps $F^k(\overline{\mathrm{C}[\overline{x}]},\overline{y})$ to $F^k(\overline{\mathrm{C}[\overline{v}]},\overline{w}) \equiv t^{\mathrm{R}}$. By the definition of the general scheme, $\overline{x} \subseteq \overline{y}$ and hence $\overline{v} \subseteq \overline{w}$. Then, since $\overline{w}$ are terms in $\{\mathrm{R}\}$, by hypothesis we have $\overline{w} \in \mathsf{Red}^{\overline{\sigma_1}}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$, where $\overline{\sigma_1} \subseteq \overline{\sigma}$. We can then infer that $\overline{v} \in \mathsf{Red}^{\overline{\sigma_2}}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$, where $\overline{\sigma_2} \subseteq \overline{\sigma_1}$.
Take a derivation for our term $F^k(\overline{\mathrm{C}[\overline{v}]},\overline{w})$. Let $\overline{\rho_{\overline{v}}}$ be the types given to $\overline{v}$ in the derivation. Now, by Subject Reduction (Theorem 2.27) it is easy to check that, for every $1 \le j \le m$ and $1 \le k' \le |\overline{\mathrm{C}}|$ it is possible to type $\mathrm{C}_{j'_k}[\overline{x}]$ from the basis $\{\overline{x{:}\rho_{\overline{v}}}\}$. We can show that R$'$ is $\overline{\mathcal{R}^\gamma}$-reducible for $\{\overline{x{:}\rho_{\overline{v}}}\}$, as follows:
Consider a derivation for the principal pair of the left hand side of the rule:

$$\overline{y{:}\mu},\ldots \vdash_{\mathcal{E}} F^k(\overline{\mathrm{C}[\overline{x}]},\overline{y}){:}\mu'$$

and assume that $\mathcal{E}(F^k) = \mu_1 {\to} \cdots {\to} \mu_n {\to} \sigma$. Note that by the definition of the general scheme, the types appearing in the principal basis for the arguments $\overline{y}$ of $F^k$ are the types required by $\mathcal{E}(F^k)$. All the types $\overline{\tau}$ used for $\overline{x} \subseteq \overline{y}$ in this derivation satisfy $\overline{\mu} \le \overline{\tau}$. Any valid derivation for an instance of the left-hand side can be obtained by a chain of operations applied to the principal one, in particular the derivation that assigns the types $\overline{\rho}$ to the occurrences of $\overline{x}$ in $\overline{\mathrm{C}[\overline{x}]}$ and $\overline{\sigma_2}$ to the occurrences of $\overline{x}$ in $\overline{y}$. Hence, there is a chain of operations that transforms $\overline{\mu}$ into $\overline{\sigma_2}$ and $\overline{\tau}$ into $\overline{\rho}$. Recall that substitution and expansions preserve the $\le$ relation on types. Lifting and closure can transform $\overline{\mu}$ or its instances into smaller types, hence, still smaller than $\overline{\tau}$. In this way we can build a derivation for $F^k(\overline{\mathrm{C}[\overline{v}]},\overline{w})$ where the types $\overline{\rho_{\overline{v}}}$ assigned to $\overline{v}$ are such that $\overline{\sigma_2 \le \rho_{\overline{v}}}$. Now, by Lemma 3.6, we have $\overline{v} \in \mathsf{Red}^{\overline{\rho_{\overline{v}}}}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$, that is R$'$ is $\overline{\mathcal{R}^\gamma}$-reducible for $\{\overline{x{:}\rho_{\overline{v}}}\}$[1].

[1] With the version of the scheme that does not require $\overline{v} \subseteq \overline{w}$ but assumes that the terms in $\overline{v}$ that do not appear in $\overline{w}$ are assigned base types, we can deduce that R$'$ is reducible because $\mathcal{SN}(\overline{v})$.

For every $1 \le j \le m$, $F^k$ does not occur in $\overline{\mathsf{C}[j]}$ (by definition of the general scheme), hence for every $1 \le k' \le |\overline{\mathsf{C}[j]}|$, $\mathcal{I}(F^k(z_1,\dots,z_n)^{\mathrm{R}}) \gg_1^{SN} \mathcal{I}(\mathsf{C}_{j'_k}[\overline{x}]^{\mathrm{R}'})$. Now, since $\mathrm{R}'$ is $\overline{\mathcal{R}^\gamma}$-reducible for $\{\overline{x{:}\rho_{\overline{v}}}\}$, it is possible to apply the induction hypothesis obtaining $\mathsf{C}_{j'_k}[\overline{x}]^{\mathrm{R}'} \in \mathsf{Red}^{\tau_{jk'}}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$ for any type $\tau_{j_{k'}}$ we can give to $\mathsf{C}_{j'_k}[\overline{x}]$ from the basis $\{\overline{x{:}\rho_{\overline{v}}}\}$. In particular when $\tau_{j_1}\to\dots\to\tau_{j_n}\to\hat{\sigma}_j$ is the type used for $F^k$ in the sub-derivation for the right hand side of our reduction rule having $F^k(\overline{\mathsf{C}_j[\overline{x}]},\overline{w})$ as the subject of the conclusion.

$(Step\ II)$: Let, for $1 \le j \le m$, $\mathrm{R}_j$ be the term-substitution such that

$$F^k(z_1,\dots,z_c,\overline{y})^{\mathrm{R}_j} \equiv F^k(\overline{\mathsf{C}_j[\overline{v}]},\overline{w}) \equiv F^k(\overline{\mathsf{C}_j[\overline{x}]},\overline{w})^{\mathrm{R}'}$$

$(c = |\overline{\mathsf{C}}|)$. By $(Step\ I)$, $\mathrm{R}_j$ is $\overline{\mathcal{R}^\gamma}$-reducible for

$$B' = \{z_1{:}\tau_{j_1},\dots,z_{c_j}{:}\tau_{j_{c_j}},\overline{y{:}\sigma_1}\},$$

where $\overline{\sigma_1} \subseteq \overline{\sigma}$. Since $\overline{\mathsf{C}} \rhd_{mul} \overline{\mathsf{C}[j]}$, and $\rhd$ is closed under term-substitution, also $\overline{\mathsf{C}}^{\mathrm{R}'} \rhd_{mul} \overline{\mathsf{C}[j]}^{\mathrm{R}'}$. So $\mathcal{I}(F^k(z_1,\dots,z_n)^{\mathrm{R}}) \gg_3^{SN} \mathcal{I}(F^k(z_1,\dots,z_{c_j},\overline{y})^{\mathrm{R}_j})$, and therefore, by induction, $F^k(z_1,\dots,z_{c_j},\overline{w})^{\mathrm{R}_j} \in \mathsf{Red}^{\hat{\sigma}_j}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$.

$(Step\ III)$: Let $\hat{v}$ be the term obtained by replacing, in the right-hand side of the rule, the terms $F^k(\overline{\mathsf{C}1\overline{v}},\overline{w}),\dots,F^k(\overline{\mathsf{C}_m[\overline{v}]},\overline{w})$ by fresh variables $z'_1,\dots,z'_m$, that is, $\hat{v} = \mathsf{C}'[z'_1,\dots,z'_m,\overline{y}]$. Let $\mathrm{R}''$ be the term-substitution such that

$$\hat{v}^{\mathrm{R}''} \equiv \mathsf{C}'[F^k(\overline{\mathsf{C}1\overline{v}},\overline{w}),\dots,F^k(\overline{\mathsf{C}_m[\overline{v}]},\overline{w}),\overline{w}],$$

then $t^{\mathrm{R}} \to \hat{v}^{\mathrm{R}''}$. Notice that, by $(Step\ II)$, $\mathrm{R}''$ is $\mathcal{R}^\gamma$-reducible for the basis $B'' = \{z'_1{:}\hat{\sigma}_1,\dots,z'_m{:}\hat{\sigma}_m,\overline{y{:}\sigma_1}\}$, where $\overline{\sigma_1} \subseteq \overline{\sigma}$. When an $F^j$ occurs in $\hat{v}$ then, by the general scheme, $j < k$ and, therefore, $\mathcal{I}(F^k(z_1,\dots,z_n)^{\mathrm{R}}) \gg_1^{SN} \mathcal{I}(\hat{v}^{\mathrm{R}''})$. Hence, $\hat{v}^{\mathrm{R}''} \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$, by induction. Since $t' \equiv \hat{v}^{\mathrm{R}''}$, we get $t' \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$.

$C)$ $t \equiv Ap(z_1,z_2)$ where $z_1,z_2 \in \mathcal{X}$. We prove this part by induction on the structure of the derivation for $B \vdash_{\mathcal{E}} Ap(z_1,z_2){:}\sigma$.

$(\le),(\mathcal{F})$: Not applicable.

$(\to E)$: Then there is a $\tau$ such that $B \vdash_{\mathcal{E}} z_1{:}\tau\to\sigma$ and $B \vdash_{\mathcal{E}} z_2{:}\tau$. Then, by rule $(\le)$, there are $\rho_1,\rho_2$ such that $\{z_1{:}\rho_1,z_2{:}\rho_2\} \subseteq B$, $\rho_1 \le \tau\to\sigma$, and $\rho_2 \le \tau$. Since $z_1{}^{\mathrm{R}}$ and $z_2{}^{\mathrm{R}}$ are $\overline{\mathcal{R}^\gamma}$-reducible in $B$, we have $z_1{}^{\mathrm{R}} \in \mathsf{Red}^{\rho_1}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$, and $z_2{}^{\mathrm{R}} \in \mathsf{Red}^{\rho_2}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$. Then, by Lemma 3.6, also $z_1{}^{\mathrm{R}} \in \mathsf{Red}^{\tau\to\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$, and $z_2{}^{\mathrm{R}} \in \mathsf{Red}^{\tau}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$. Then, by Definition 3.3, $Ap(z_1{}^{\mathrm{R}},z_2{}^{\mathrm{R}}) \in \mathsf{Red}^{\sigma}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}]$. Notice that $Ap(z_1{}^{\mathrm{R}},z_2{}^{\mathrm{R}})$ is the same as $Ap(z_1,z_2)^{\mathrm{R}}$.

$(\cap I)$: Then $\sigma \equiv \tau_1\cap\dots\cap\tau_m$. The thesis follows easily by induction, since

$$\mathsf{Red}^{\tau_1\cap\dots\cap\tau_m}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}] \equiv \mathsf{Red}^{\tau_1}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}] \cap \dots \cap \mathsf{Red}^{\tau_m}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}].$$

$(\forall E)$: Then $\sigma \equiv \sigma'[\tau/\varphi']$ where the type in the premise of the rule is $\forall\varphi'.\sigma'$. Since $\forall\varphi'.\sigma' \le \sigma'[\tau/\varphi']$, the result follows by induction and Lemma 3.6.

$(\forall I)$: Then $\sigma \equiv \forall\varphi'.\sigma'$ and $B \vdash_{\mathcal{E}} t{:}\sigma'$. Moreover, $\varphi' \notin FV(\sigma_1,\dots,\sigma_n)$. By induction, for any sequence of types $\overline{\gamma},\delta$ such that $|\overline{\gamma},\delta| = |\overline{\varphi},\varphi'|$ and any sequence of reducibility candidates $\overline{\mathcal{R}^\gamma},\mathcal{S}^\delta$: if R is a term-substitution $\overline{\mathcal{R}^\gamma},\mathcal{S}^\delta$-reducible for $B$, then $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma'}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathcal{S}^\delta/\varphi']$. Since $\varphi' \notin FV(\sigma_1,\dots,\sigma_n)$, it follows that, for any $1 \le j \le n$,

$$\mathsf{Red}^{\sigma_j}[\overline{\mathcal{R}^\gamma}/\overline{\varphi},\mathcal{S}^\delta/\varphi'] \equiv \mathsf{Red}^{\sigma_j}[\overline{\mathcal{R}^\gamma}/\overline{\varphi}].$$

This means that a term-substitution which is $\overline{\mathcal{R}^{\gamma}},\mathcal{S}^{\delta}$-reducible for $B$, is also $\overline{\mathcal{R}^{\gamma}}$-reducible for $B$. Hence we can restate the induction hypothesis as follows: for any sequence of types $\overline{\gamma}$ such that $|\overline{\gamma}| = |\overline{\varphi}|$ and any sequence of reducibility candidates $\overline{\mathcal{R}^{\gamma}}$, if R is a term-substitution $\overline{\mathcal{R}^{\gamma}}$-reducible for $B$, then for any type $\delta$ and reducibility candidate $\mathcal{S}^{\delta}$, $t^{\mathrm{R}} \in \mathsf{Red}^{\sigma'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi},\mathcal{S}^{\delta}/\varphi']$. By definition of reducibility candidates this means that $Ap(z_1,z_2)^{\mathrm{R}} \in \mathsf{Red}^{\forall\varphi'.\sigma'}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$.

*ii*) $t$ is not neutral. Let $t \equiv \lambda x.u$.

We can proceed by induction on the structure of the derivation of $B \vdash_{\mathcal{E}} \lambda x.u{:}\sigma$. The argument is similar to that used for the case $t \equiv Ap(z_1,z_2)$. We prove only the interesting part, when the last rule is $(\rightarrow I)$, and $\sigma \equiv \rho\rightarrow\beta$. Then we need to show that, given $v$ such that $v \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$, we have that $Ap(t^{\mathrm{R}},v) \in \mathsf{Red}^{\beta}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Since the term $Ap(t^{\mathrm{R}},v)$ is neutral, by (*C3*) it is enough to prove $t' \in \mathsf{Red}^{\beta}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$ for all $t'$ such that $Ap(t^{\mathrm{R}},v) \rightarrow t'$. This will be proved by induction on the sum of the length of the rewrite sequences out of $v$ and out of R. Note that since $v$ and R are reducible, by (*C1*) $\mathcal{SN}(v)$ and $\mathcal{SN}(R)$.

(*Base*): If $v$ and R are in normal form, the only reduction step out of $Ap(t^{\mathrm{R}},v)$ can be:

$$Ap((\lambda x.u)^{\mathrm{R}},v) \rightarrow t' \equiv u^{\mathrm{R}'},$$

where $\mathrm{R}' \equiv \mathrm{R} \cup \{x \mapsto v\}$. $\mathrm{R}'$ is reducible because $v \in \mathsf{Red}^{\rho}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Now, since $(\lambda x.u) \trianglerighteq u$, $\mathcal{I}(\lambda x.u^{\mathrm{R}}) \gg_2^{SN} \mathcal{I}(u^{\mathrm{R}'})$. By induction we have $u^{\mathrm{R}'} \in \mathsf{Red}^{\beta}[\overline{\mathcal{R}^{\gamma}}/\overline{\varphi}]$. Note that $u^{\mathrm{R}'} \equiv t'$.

(*Induction step*): Otherwise, the reduction step out of $Ap(t^{\mathrm{R}},v)$ must take place inside $v$ or R. Then $t'$ is computable by induction. ■