Subtyping, Recursion and Parametric Polymorphism in Kernel Fun

Dario Colazzo Giorgio Ghelli

Dipartimento d'Informatica - Università di Pisa
Via F. Buonarroti 2 - 56127 - Pisa - Italy
{*colazzo, ghelli*}*@di.unipi.it*

Abstract

We study subtype checking for recursive types in system kernel Fun, a typed λ-calculus with subtyping and bounded second-order polymorphism. Along the lines of [AC93], we define a subtype relation over kernel Fun recursive types, and prove it to be transitive. We then show that the natural extension of the algorithm introduced in [AC93] to compare first-order recursive types yields a non complete algorithm. Finally, we prove the completeness and correctness of a different algorithm, which lends itself to efficient implementations.

Key words: type theory and type systems, subtyping, recursive types, kernel Fun.

1 Introduction

1.1 Results

Recursive types are supported by all typed languages, since they are needed to define fundamental data structures, such as lists and trees, and occur in common programming patterns, such as the subject-observer [Cop].

Two different approaches to recursive types have been studied in the literature. Given a recursive definition *let rec* $X = T[X]$, the strong (or *equality based*) approach makes X equal to $T[X]$ ([Nel91, AC93]), while the weak (*isomorphism based*) approach ([GMW79, Gun92]) only gives the programmer a pair of functions $fold_{T[X]}$: $T[X] \to X$ and $unfold_{T[X]}$: $X \to T[X]$. The weak approach makes type and subtype checking very easy. The strong approach

is easier for programmers to use, but makes subtype checking much more challenging, and is the one we study in this paper.

The combination of subtyping and recursive types has a significant practical relevance. Both notions appear in every typed object-oriented language, and are even useful for the compilation of languages that do not have a subtype relation (as in the ML to JavaVM compilation project at Persimmon IT, where subtyping between strongly recursive types is used in the intermediate language for optimization purposes [Ken98, BKR98]).

In [AC93] Amadio and Cardelli studied the problem of defining and checking a subtype relation between *first-order* strongly recursive types. In that paper they defined a subtyping algorithm which is sound and complete with respect to several equivalent axiomatizations of the subtype relation.

In this paper we study the integration of strong recursion in a second-order type system with subtyping. For this purpose we refer to system *kernel Fun*, an abstract version of Fun [CW85], a language that combines subtyping with parametric polymorphism, and that allows the definition of bounded quantified types, i.e. polymorphic types whose quantifier ranges over a set of subtypes of a given type. Languages of the Fun family, with their extensions, were the first foundational tools used to model object-oriented languages with expressive and strong types (see, for example, [AC96, BCP96, DT88, FM96, Ghe91, GM94]). Although most current languages are based on some variant of type-comparison by name, the structural approach to type-comparison that has been pursued by the type-theoretical community has some advantages, especially in the context of open systems [AC93], where the structural approach allows type checking to be performed independently of the particular environment where data and programs migrate. Moreover, the study of the structural approach lays the foundations for a full understanding of comparison by name.

The main results of this paper are:

- We define a subtype relation over recursive kernel Fun, and we prove that this subtype relation is transitive; transitivity is the key property needed to prove that 'well-typed programs do not go wrong'.
- We define an algorithm to check this subtype relation, and we prove that it is correct and complete. The algorithm is not obvious, and the proof of its correctness is very challenging.
- We show that the most natural algorithm for the same problem is incomplete (Section 4.1), while its obvious generalization is incorrect (Section 4.2), even if we restrict ourselves to a limited subset of system kernel Fun.
- We show that our algorithm can be implemented in an efficient way.

Our algorithm is obtained by a non-trivial extension of the first-order Amadio-Cardelli algorithm. Their algorithm is based on the idea of keeping track of

the pairs of compared types that are met during the subtype-checking process, so that it can stop when the 'same' pair is met for the second time. We show that the obvious extension of their algorithm to kernel Fun fails to be complete when 'sameness' is generalized to α-equivalence (or, even worse, simply syntactical equality), and it fails to be correct when sameness is generalized to a quite natural notion of 'similarity'. However, we obtain a correct and complete algorithm if we generalize sameness to this similarity relation, but stop execution only when a similar pair is met for the *third* time.

In this study we do not examine the complexity of our algorithm. We know that it may have an exponential behavior [Ghe96], and we suspect it may be made polynomial, but in this paper this will remain an open issue.

1.2 Related Work

The problem of subtyping *first-order* recursive types was first addressed by Amadio and Cardelli in [AC93], where they defined a subtype relation over recursive types, and a sound and complete subtyping algorithm. In [KPS93] a more efficient subtyping algorithm is described.

In [BH97] another possible axiomatization of the subtype relation between first-order recursive types is presented, which is equivalent to the one defined in [AC93]. The main difference is that while in [BH97] the subtype relation is defined by means of deduction rules that are interpreted coinductively, in [AC93] the subtype relation is based on a notion of preorder over infinite trees representing the infinite unfolding of recursive types.

In this paper we define the subtype relation between recursive types by means of deduction rules, as in [BH97], but we generalize that work to second-order systems. This kind of definition lends itself to be transformed into a set of rules that define a subtyping checking algorithm. Moreover, the strong similarity between the rules that define the subtype relation and the rules that define the subtyping algorithm facilitates the proofs of completeness and soundness of the latter with respect to the former.

In [GLP00] an introduction to recursive types and subtyping algorithms is presented; in particular, the authors highlight the connection between coinductive structures, used to deal with recursion, and the framework of nonrecursive types and ordinary subtyping. In [AF96], first-order recursive types are studied from a syntactic perspective and the two possible approaches to recursion, *equality-based* and *isomorphism-based*, are compared. In particular, the equivalence between the two approaches is proved.

Turning now to papers dealing with the subtype relation for recursive types

in second-order systems, a negative result was given in [Ghe93]: any attempt to extend system $F_{\leq}$ [Ghe90] with recursive types leads to the definition of a non conservative extension of the system. This is due to the fact that the inductive and the co-inductive interpretation of the subtyping rules of $F_{\leq}$ yield two different relations (and this is the main reason behind the undecidability of subtyping in this system [Pie94, Ghe95]). Other papers where second-order recursive types are studied do not deal with the subtype checking problem in much depth. In [BCP96], a recursive extension of system $\mathrm{F}_{<:}\omega$ [Car90, CL91] is defined, but only as a tool to compare different models of object-oriented languages. Hence, the algorithmical aspects of the subtyping problem are not considered, and the system defined is far less powerful than our extension of the system in [AC93]. The extension of kernel Fun with recursive types is studied in [Car89] from a semantical point of view, and it is proved that extending the system with recursive types is consistent.

To our knowledge, the extended abstract of this paper published in [CG99] was the first paper to address the problem of subtype checking second-order recursive types, i.e. how to actually check that two such types are in the subtype relation. This paper extends that abstract in that:

- we present here, for the first time, the proof of all our results, which constitutes the main body of this paper; although the proof is complex, its reduction to this (relatively) manageable form was the most challenging aspect;
- we prove the transitivity of the type system we define; transitivity of subtyping is the key lemma in order to prove subject-reduction for kernel Fun terms.

Building on the results of [CG99], Alan Jeffrey [Jef01] defines a subtype relation between $F_{\leq}$ types that conservatively extends the relation we defined on recursive kernel Fun, and gives a characterization of this relation by means of polar bisimulations between labeled transition systems. Based on this bisimulation, he defines a subtyping algorithm for recursive $F_{\leq}$ that is partially correct: if it terminates it returns the right answer. As implied by the non-conservativity theorem of [Ghe93], Jeffrey's system is not a conservative extension of $F_{\leq}$, and indeed it terminates with success over $F_{\leq}$ judgements that are not provable in non-recursive $F_{\leq}$. In the special case of kernel Fun recursive types, Jeffrey's algorithm is correct and complete. Jeffrey's algorithm is more general than ours, since it deals with a subset of $F_{\leq}$ that is strictly larger than kernel Fun, but is based on variable renaming and α-conversion. Our algorithm avoids variable renaming and α-conversion, and this property, as we explain later, is extremely important for its performance.

A different strand of research deals with a notion of subtyping where an instance of a polymorphic type is a supertype of the type itself. In such a system,

the subtyping problem for second-order languages is undecidable even when quantification is unbounded [TU96]. We do not comment on papers in this family since their results and techniques cannot be applied to the subtype checking of languages in the Fun family.

1.3 Structure of the Paper

The paper is organized as follows. In Section 2 we present a recursive version of system kernel Fun. We extend the Amadio-Cardelli first-order subtype system to system kernel Fun, adopting the style of [BH97]: we present a set of rules whose coinductive interpretation defines the subtype relation. In Section 3 we introduce a new version of the coinductive rules where every type occurrence is labeled. These labels are used both to rename types in a way that makes it easier to test for type 'similarity', and also to reason about the properties of the proofs in this type system. In Section 4 we first prove that the most natural algorithm to solve the subtype checking problem is not complete, then we prove that an algorithm that stops when a 'similar' pair is met for the second time is not sound. Finally, we define our algorithm, which stops the third time a similar pair is met.

In Section 5 we prove the completeness and soundness of our algorithm. In Section 6 we prove that the subtype relation is transitive. Finally, in Section 7 we outline our conclusions and areas of future research.

2 Recursive Kernel Fun

2.1 Syntax

Here we only present the kernel Fun types, since the type and reduction rules for terms are not affected by the introduction of recursion; for a complete introduction to the system see [CW85, Ghe90, CG92, CMMS94].

We will use lower case letters ($t,\ u,\ v, \ldots$) to indicate type variables, and upper case letters ($X,\ Y,\ K, \ldots$) for recursion variables. Types are defined as follows:

$$\textbf{Types} \qquad T, U ::= \top \mid t \mid X \mid \mu X.\forall t \leq T.U \mid \mu X.T \to U$$

$\top$ is the top type, the supertype of all types. $\mu X.\forall t \leq T.U$ is a universally quantified type where the type variable t ranges over the subtypes of T and is

bound in U. $\mu X.T \to U$ denotes the type of all functions from values of type T to values of type U.

In a type $\mu X.\forall t \leq T.U$, or $\mu X.T \to U$, an occurrence of X in T or in U recursively denotes the whole type, $\mu X.\forall t \leq T.U$, or $\mu X.T \to U$, respectively. This means for example that, in any interpretation that respects this intuition, the following equations must hold [1] :

$$\mu X.T = [\mu X.T/X]\,T = [[\mu X.T/X]\,T/X]\,T \ldots$$

The notation $[U/X]\,T$ is standard and indicates capture-free variable substitution.

We consider a grammar where only function types or bounded quantified types can be the body of a recursive type; as an alternative we may only add $\mu X.T$ to kernel Fun type language, with formation rules that forbid empty types such as $\mu X.X$. This is mainly a stylistic choice, with no major effect on either the power of the language or the difficulty of the problem.

Notation 2.1 *Hereafter $\forall t.\,T$ will be used as an abbreviation for $\forall t \leq \top.\,T$.*

In type theory, two types that only differ in the names of their bound variables (α-equivalent types) are usually identified. We will consider such types as different (although one will be a subtype of another), because variable names play a central role in our subtyping algorithm. Moreover, we require that all variables in a single type have different names, and we say that a type that satisfies this condition is an "R-Type" (well-formed *Recursive Type).* This condition is not restrictive with respect to the usual presentation of the system, where types are interpreted modulo α-equivalence, since every type is α-equivalent to an R-Type. Variable uniqueness is enforced to guarantee some good formation properties during subtype checking (Section 4.4) and to simplify the definition of the termination conditions of our subtyping algorithm (Section 4.3).

Definition 2.2 *A type belongs to* R-Types *if and only if all its variables have different names.*

Definition 2.3 *On R-Types we define* $\mathrm{DV}(T)$ *and* $\mathrm{FV}(T)$*, the defined and*

[1] With a notational abuse, we will also use T as a metavariable for the body $\forall t \leq T.U$ or $T \to U$ of a recursive type.

free variables of T respectively, as follows:

$$
\begin{array}{ll}
\mathrm{DV}(\top) & = \emptyset \\
\mathrm{DV}(t) & = \emptyset \\
\mathrm{DV}(X) & = \emptyset \\
\mathrm{DV}(\mu X.\forall t \leq T'.U') & = \{X, t\} \cup \mathrm{DV}(T') \cup \mathrm{DV}(U') \\
\mathrm{DV}(\mu X.T' \to U') & = \{X\} \cup \mathrm{DV}(T') \cup \mathrm{DV}(U')
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{FV}(\top) & = \emptyset \\
\mathrm{FV}(t) & = \{t\} \\
\mathrm{FV}(X) & = \{X\} \\
\mathrm{FV}(\mu X.T' \to U') & = (\mathrm{FV}(T') \cup \mathrm{FV}(U')) - \{X\} \\
\mathrm{FV}(\mu X.\forall t \leq T'.U') & = (\mathrm{FV}(T') \cup (\mathrm{FV}(U') - \{t\})) - \{X\}
\end{array}
$$

The syntax of our system also includes the following definitions.

Pre-Judgements	$P ::= \Gamma \rhd \mathrm{Env} \mid \Gamma \rhd T\ \mathrm{Type} \mid \Pi \rhd T \leq U$
Judgements	$J ::= \Gamma \vdash \mathrm{Env} \mid \Gamma \vdash T\ \mathrm{Type} \mid \Pi \vdash T \leq U$
Bi-Environments	$\Pi ::= () \mid \Pi, (t, u) \leq (T, U) \mid \Pi, (X = T, Y = U)$
Environments	$\Gamma ::= () \mid \Gamma, t \leq T \mid \Gamma, X = T$

Following [Ghe96], in our system we distinguish between *pre-judgements* and *judgements.* Pre-judgements represent the input for the subtype and good formation checking process; a judgement indicates that the corresponding pre-judgement holds.

The main judgement is $\Pi \vdash T \leq U$, stating that, with respect to the *bi-environment* Π, T is a subtype of U. The corresponding pre-judgement is $\Pi \rhd T \leq U$.

In $\Pi \rhd T \leq U$ and $\Pi \vdash T \leq U$ the environment Π defines (i.e. binds) type and recursion variables occurring free in T and U. In particular, it contains assumptions of shape $(t, u) \leq (T', U')$. Given $\Pi \rhd T \leq U$, an assumption $(t, u) \leq (T', U')$ in Π defines t in T and u in U. Moreover, the assumption implies that t is a subtype of u and u is a subtype of t, and indicates that T' and U' are the bounds for t and u, respectively.

The use of bi-environments in subtyping judgement is another feature we borrow from [Ghe96]. They are adopted to reduce the need for variable renaming during the subtype checking process. For example, the pre-judgement $() \rhd (\forall t \leq \top.t) \ \leq \ (\forall u \leq \top.u)$ is proved by reducing it to

$$(t, u) \leq (\top, \top) \rhd t \leq u$$

where t and u are unified in the environment, without renaming them with a common name. This allows us to keep the original name of type variables during subtyping checking, thus easing similarity checking (to be defined later) among the generated pre-judgements. As already stated, similarity checking is crucial in our subtyping algorithm.

In $\Pi \rhd T \leq U$ and $\Pi \vdash T \leq U$, recursive variable definitions in Π have the form $(X = T', Y = U')$, stating that each free occurrence of X in T (Y in U) stands for the type T' (the type U'). In this case no relationship is implied between X and Y; definitions are paired because we only compare one recursive type with another one.

A bi-environment Π can be seen as the composition of two distinct environments, one for each of the two compared types. For this reason we call it a *bi-environment*, and use the traditional term *environment* for environments Γ, which define variables one at a time. Environments appear in type well-formation judgements $\Gamma \vdash T$ Type, which essentially state that every variable in T is defined and that, if the definition of a variable is in the scope of another one, then their names differ.

The two environments into which a bi-environment Π can be divided are respectively denoted by Left(Π) and Right(Π), defined in 2.4. We also use the notation $\Pi \vdash T, U$ Type as a shortcut for Left(Π) $\vdash$ T Type $\wedge$ Right(Π) $\vdash$ U Type.

Definition 2.4 *We consider the following operations over bi-environments. We define them on bi-environment elements. They are lifted to the whole bi-environment in the obvious element-wise way.*

	$(t, t') \leq (T, T')$	$(X = T, Y = T')$
Def	(t, t')	(X, Y)
Swap	$(t', t) \leq (T', T)$	$(Y = T', X = T)$
Left	$t \leq T$	$X = T$
Right	$t' \leq T'$	$Y = T'$

where (t, t') and (X, Y) are ordered pairs of variables.

These bi-environment operations will be used to define good formation and subtyping rules in the next two sections. Moreover, we will use the following operators on environments.

Definition 2.5 $\Gamma(t)$ *indicates the bound of t in Γ, i.e. the type T such that $t \leq T \in \Gamma$.*

Definition 2.6 *On environments* Γ, $\mathrm{Def}(\Gamma)$ *denotes the set of variables defined in* Γ. $\mathrm{Def}(\Gamma)$ *is defined in the following way*

$$\begin{aligned}
\mathrm{Def}() &= \emptyset \\
\mathrm{Def}(\Gamma,\ t \leq T) &= \mathrm{Def}(\Gamma) \cup \{t\} \\
\mathrm{Def}(\Gamma,\ X = T) &= \mathrm{Def}(\Gamma) \cup \{X\}
\end{aligned}$$

2.2 Good Formation

The good formation rules we present strictly resemble standard ones [Ghe96, Ghe90]. The differences are that there are new rules for recursion plus a notion of well-formedness, which we borrow from [Ghe96], which is stronger, in some sense, than the traditional one — in rule (T-VarForm), as discussed below.

2.2.0.1 Good formation rules

$$\frac{}{() \vdash \mathrm{Env}} \quad (\Gamma \text{ EmptyForm}) \qquad \frac{t \notin \mathrm{Def}(\Gamma) \quad \Gamma \vdash T\ \mathrm{Type}}{\Gamma, t \leq T \vdash \mathrm{Env}} \quad (\Gamma \text{ BoundForm})$$

$$\frac{X \notin \mathrm{Def}(\Gamma) \quad \Gamma \vdash T\ \mathrm{Type}}{\Gamma, X = T \vdash \mathrm{Env}} \quad (\Gamma \text{ EqForm})$$

$$\frac{t \leq T \in \Gamma \quad \Gamma \vdash T\ \mathrm{Type}}{\Gamma \vdash t\ \mathrm{Type}} \quad (\text{T-VarForm})$$

$$\frac{\Gamma \vdash \mathrm{Env}}{\Gamma \vdash \top\ \mathrm{Type}} \quad (\top \text{ Form}) \qquad \frac{X \in \mathrm{Def}(\Gamma) \quad \Gamma \vdash \mathrm{Env}}{\Gamma \vdash X\ \mathrm{Type}} \quad (\text{R-VarForm})$$

$$\frac{\Gamma, X = \top \vdash T\ \mathrm{Type} \quad \Gamma, X = \top \vdash U\ \mathrm{Type}}{\Gamma \vdash \mu X.T \to U\ \mathrm{Type}} \quad (\to \text{ Form})$$

$$\frac{\Gamma, X = \top, t \leq T \vdash U\ \mathrm{Type}}{\Gamma \vdash \mu X.\forall t \leq T.U\ \mathrm{Type}} \quad (\forall \text{ Form})$$

Note the particular definition of (T-VarForm). In the standard presentation of the system, variable good formation is defined by the following rule:

$$\frac{\Gamma \vdash \mathrm{Env} \quad t \in \mathrm{Def}(\Gamma)}{\Gamma \vdash t\ \mathrm{Type}} \quad (\text{WeakVarForm})$$

Here, we adopt a stronger good formation property. Later (Property 4.14), we show that when this stronger well-formedness is satisfied by a pre-judgement, then all pre-judgements created during the subtype-checking process are still

well-formed, without any need for renaming; this is not true with the traditional weak notion of well-formedness. For example, consider the judgement $() \vdash T \leq U$ with $T = \forall t \leq (\forall u.u).\forall u.t$ and $U = \forall t' \leq (\forall u'.u').\forall s.(\forall v.v)$. We have $() \vdash T \leq U$ Type if (WeakVarForm) is used instead of (T-VarForm), but $() \vdash T \leq U$ is reduced in three steps to the ill-formed judgement $(t, t') \leq \ldots, (u, s) \leq (\top, \top) \vdash \forall u.u \leq \forall v.v$. This judgement is ill-formed because the type variable u is defined twice, once in the environment and once in the comparison. In any case, the stronger notion of good formation we adopt is not restrictive, since every weakly well-formed pre-judgement has a corresponding α-equivalent strongly well-formed judgement.

The combined use of bi-environments, R-Types, and strong well-formedness will allow us to confine the use of variable renaming to subtyping rules that unfold recursive types.

Regarding rules ($\rightarrow$ Form) and ($\forall$ Form) note that, in their premises, the environment Γ is enriched with $X = \top$ instead of $X = \mu X.\forall t \leq T.U$ (or $X = \mu X.T \rightarrow U$). So we just extend Γ with some information telling us that the environment defines the recursion variable X. This is the only information needed to check good formation.

It is not difficult to prove that these rules define a terminating good formation checking algorithm; the proof is essentially the same as the termination proof of kernel Fun subtype checking (see [Ghe95, KS92]).

2.3 Subtyping

In [AC93], a recursive type has been defined as a subtype of another if the possibly infinite subtype comparison among their unfoldings does not fail. This idea can be formalized in different ways (see [BH97, AC93]); in this paper we formalize it by means of a set of coinductive subtyping rules, which essentially consists of the standard algorithmical rules for kernel Fun [CW85, Ghe96] enriched with rules to deal with recursion. Before presenting them, we make a brief formal digression on inductive and coinductive subtyping (see [GLP00] for a broader discussion on this topic).

Given a set of subtyping rules S, the well-formedness relation

$$\mathcal{WF} = \{(\Pi, T, U) : \Pi \vdash T, U \text{ Type}\}$$

and a generating function F_S defined on subsets R of $\mathcal{WF}$ in the following

way

$$F_S(R) = \{(\Pi, T, U) \mid \Pi \vdash T \leq U \text{ is a conclusion of a ground instance } r \text{ of a rule in } S \text{ and all the premises of } r \text{ are in } R\}$$

we say that a relation $R \subseteq \mathcal{WF}$ is *compatible* with S if and only if R is a fixed point of F_S, i.e. $R = F_S(R)$.

The existence of both least and greatest fixed points, denoted by μF_S and νF_S respectively, is guaranteed by Knaster-Tarski theorem [Tar55], provided that F_S is monotone:

$$R \subseteq R' \Rightarrow F_S(R) \subseteq F_S(R')$$

This will be the case for the subtyping rules we are going to introduce.

The least fixed point (inductive subtyping) only contains subtyping judgements with a finite proof, while the greatest fixed point (coinductive subtyping) may also contain judgements with an infinite proof. Our subtype relation is defined by the greatest fixed point and contains such judgements.

The Knaster-Tarski theorem provides a way to construct the greatest fixed point as the limit:

$$\nu F_S = \bigcap_{i=1}^{\infty} F_S^i(\mathcal{WF})$$

and this means that νF_S can be constructed by starting from the full relation $\mathcal{WF}$ and by eliminating, at each stage i, all pre-judgements that cannot be deduced by F_S. The pre-judgements that are not eliminated at any stage are in νF_S.

We may also express this fact by saying that a pre-judgement $\Pi \rhd T \leq U$ holds — $(\Pi, T, U) \in \nu F_S$ — if either a finite or an infinite proof for it exists. This is the interpretation of subtyping that we will adopt here (see Definition 3.25 for the formal definition). Of course this is not an "algorithmic" definition.

The set of subtyping rules we consider is defined below. We assume some familiarity with kernel Fun and subtyping recursive types [CW85, AC93, GLP00].

Notation 2.7 (Type equality: $T \triangleq U$) *We define:*

$$\Pi \vdash T \triangleq U \;\Leftrightarrow_{def}\; \Pi \vdash T \leq U \;\wedge\; \mathrm{Swap}(\Pi) \vdash U \leq T$$

Notation 2.8 (Fresh renaming: $\Pi \vdash T\!\uparrow\! \leq U$ and $\Pi \vdash T \leq U\!\uparrow$) *In the following, $\Pi \vdash T\!\uparrow\! \leq U$ means that there exists an R-type T' which is α-*

equivalent to T, such that no variable bound in T' is bound in $\mathrm{Left}(\Pi)$*, and such that* $\Pi \vdash T' \leq U$*. The definition of* $\Pi \vdash T \leq U\!\uparrow$ *is analogous.*

Equivalently, we can say that $T\!\uparrow$ denotes an arbitrary R-type obtained from T by renaming its bound variables with variables that are fresh w.r.t. $\mathrm{Left}(\Pi)$. Inspection of the type rules shows that the specific choice of the fresh variables is irrelevant.

2.3.0.2 Subtyping rules

$$\frac{\Pi \vdash T, \top \text{ Type}}{\Pi \vdash T \leq \top}\ (\top_{\leq}) \qquad \frac{(t,u) \in \mathrm{Def}(\Pi) \quad \Pi \vdash t, u \text{ Type}}{\Pi \vdash t \leq u}\ (\mathrm{Id}_{\leq})$$

$$\frac{\begin{array}{l}(t,u) \leq (T',U') \in \Pi \\ \textit{for all } X.\ (U \neq X) \quad U \neq \top \quad U \neq u \quad \Pi \vdash T' \leq U\end{array}}{\Pi \vdash t \leq U}\ (\mathrm{VarTrans}_{\leq})$$

$$\frac{\begin{array}{l}\Pi' = (\Pi, (X{=}\mu X.T \to U,\ Y{=}\mu Y.T' \to U')) \\ \mathrm{Swap}(\Pi') \vdash T' \leq T \quad \Pi' \vdash U \leq U'\end{array}}{\Pi \vdash \mu X.T \to U \leq \mu Y.T' \to U'}\ (\to_{\leq})$$

$$\frac{\begin{array}{l}\Pi' = (\Pi, (X{=}\mu X.\forall t{\leq}T.U,\ Y{=}\mu Y.\forall t'{\leq}T'.U')) \\ \Pi' \vdash T \mathrel{\mathring{=}} T' \quad \Pi', (t,t') \leq (T,T') \vdash U \leq U'\end{array}}{\Pi \vdash \mu X.\forall t{\leq}T.U \leq \mu Y.\forall t'{\leq}T'.U'}\ (\forall_{\leq})$$

$$\frac{X{=}T \in \mathrm{Left}(\Pi) \quad \Pi \vdash T\!\uparrow \leq U}{\Pi \vdash X \leq U}\ (\mathrm{LUnf}_{\leq})$$

$$\frac{\textit{for all } X.\ (T \neq X) \quad Y{=}U \in \mathrm{Right}(\Pi) \quad \Pi \vdash T \leq U\!\uparrow}{\Pi \vdash T \leq Y}\ (\mathrm{RUnf}_{\leq})$$

Since we are interested in solving the subtype-checking problem, we will comment on the rules with respect to their backward reading (where the problem of proving the conclusion is reduced to the problem of proving the premises). Of course, the forward reading of the same rules makes perfect sense as well.

Reflexivity of subtyping follows by rule $(\mathrm{Id}_{\leq})$ and structural induction on the type structure. (In this context, reflexivity means that $\mathrm{Left}(\Pi) = \mathrm{Right}(\Pi)$ and $\Pi \vdash T, T$ Type imply that $\Pi \vdash T \leq T$.) Moreover, instead of a general

transitive rule

$$\frac{\begin{array}{c}\mathrm{Left}(\Pi') = \mathrm{Left}(\Pi) \\ \mathrm{Right}(\Pi') = \mathrm{Left}(\Pi'') \\ \mathrm{Right}(\Pi'') = \mathrm{Right}(\Pi) \\ \Pi' \vdash T \leq V \quad \Pi'' \vdash V \leq U\end{array}}{\Pi \vdash T \leq U} \ (\mathrm{Trans}_{\leq})$$

we only impose transitivity in the specific case $t \leq U$ (rule ($\mathrm{VarTrans}_{\leq}$)). This is necessary since the greatest fixed point of a set of rules that includes transitivity is the total relation (see [GLP00]). We will prove later (Section 6) that the ($\mathrm{VarTrans}_{\leq}$) rule is sufficient to make our system transitive.

When quantified types are compared, kernel Fun requires equality of bound types and, with this restriction, the subtyping algorithm for non recursive types always terminates [Ghe96]. In our definition of the ($\forall_{\leq}$) rule, equality of bound types is expressed in the premises by the mutual subtyping judgement $\Pi' \vdash T \stackrel{\circ}{=} T'$.

In system $F_{\leq}$ [Ghe90] the inclusion between quantified types is checked by the rule:

$$\frac{\Gamma \vdash T' \leq T \quad \Gamma, t' \leq T' \vdash [t'/t]U \leq U'}{\Gamma \vdash \forall t \leq T.U \leq \forall t' \leq T'.U'} \ (\forall_{\leq})_{F_{\leq}}$$

where the equality of bounds required in kernel Fun is relaxed to a less restrictive subtyping requirement. As shown in [Ghe95], the existence of such a rule makes the $F_{\leq}$ subtyping algorithm diverge when particular pairs of types are compared. Moreover, in [Ghe93] it is shown that the existence of such divergent pre-judgements makes it difficult to extend $F_{\leq}$ with recursive types: the extended system is not conservative with respect to $F_{\leq}$, that is, there are some non-provable judgements in $F_{\leq}$ that become provable in $F_{\leq}$ enriched with recursion. For this reason, in this paper we restrict our attention to kernel Fun.

In order to guarantee well-formedness, in the premises of rules ($\forall_{\leq}$) and ($\rightarrow_{\leq}$) the bi-environment is extended with the equation defining the types occurring in the conclusion. Moreover, the ($\forall_{\leq}$) rule unifies the type variables t and t' of the two compared types.

The symbol $T\uparrow$ in ($\_\mathrm{Unf}_{\leq}$) rules denotes a copy of T where every bound variable t or Y is renamed with an arbitrary fresh name (Notation 2.8). This renaming cannot, in general, be avoided since otherwise the backward application of the rules would produce ill-formed pre-judgements. As an example, consider a comparison involving the type $\mu X.\forall t \leq \top.t \rightarrow X$. After applying rules ($\forall_{\leq}$)-($\rightarrow_{\leq}$)-($\_\mathrm{Unf}_{\leq}$)-($\forall_{\leq}$) we would end up with two definitions for X and t in the

bi-environment. In the next section we will define a systematic way to perform the renaming $T\uparrow$.

Backward rule application is deterministic: as it is easy to check, determinism is guaranteed by inequality conditions stated in rules ($\text{VarTrans}_{\leq}$) and ($\text{RUnf}_{\leq}$); in rules ($\text{VarTrans}_{\leq}$) and ($\text{RUnf}_{\leq}$), the statement *for all* X. $(T \neq X)$ means that T is not a recursion variable. Rule determinism implies that there is a unique proof tree (modulo α-equivalence) for every judgement.

The rules also define a reduction system on pre-judgements. $P \xrightarrow{(R)} P'$ indicates that P is reduced to P' by backward application of a rule called (R), i.e. P' is one of the premises, and P is the conclusion, of a ground instance of (R). $P \to P'$ means that $P \xrightarrow{(R)} P'$ for some (R). The symbol $\twoheadrightarrow$ indicates the reflexive and transitive closure of $\to$.

By rule determinism, $\Pi \vdash T \leq U$ if and only if $\Pi \rhd T \leq U$ is never reduced to a pre-judgement $\Pi' \rhd T' \leq U'$ that does not match the conclusion of any rule. However, this criterion does not define a subtyping algorithm since the unfolding rules ($\_\text{Unf}_{\leq}$) make the system diverge whenever, for example, two equal recursive types are compared. In Section 4.3 we see how this divergence can be stopped.

To simplify our study we will restrict ourselves to prejudgements P that satisfy the following conditions: exist T' and U' such that $() \rhd T' \leq U' = P$ or $() \rhd T' \leq U' \twoheadrightarrow P$, and:

- T' and U' are closed types
- $\text{DV}(T') \cap \text{DV}(U') = \emptyset$.

3 Labeled Recursive Kernel Fun

In this section we introduce a labeled variant of recursive kernel Fun that we use as a bridge between the "official" system and the algorithm we are going to present. The labeled system is based on a set of labeled types defined below, where α and β range over *paths* (elements of $\{0,1\}^*$). Every variable t_α, X_α will contain, in its name, an occurrence label α that makes it unique in the unfolding; moreover every leaf $\top$, t_α, and X_α of the type will also be marked by its occurrence β, indicated as $\top_\beta$, $t_{\alpha|\beta}$, $X_{\alpha|\beta}$. The full grammar is:

$$T, U ::= \top_\beta \mid t_{\alpha|\beta} \mid X_{\alpha|\beta} \mid \mu X_\alpha.\forall t_\alpha \leq T.U \mid \mu X_\alpha.T \to U$$

In Section 3.1 we give the formal definition of labels and we define a mapping from non-labeled types to labeled types. In Section 3.2 we define the subtype

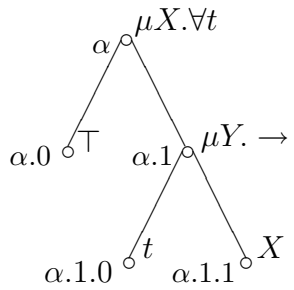
relation over labeled recursive types.

3.1 Adding Labels

To move toward our subtype algorithm, in this section we define a specific variable renaming technique to be used in the unfolding rules. Informally, we interpret the repeated backward application of the subtyping rules starting from $() \rhd T \leq U$, with T and U closed types, as a descent along the infinite unfolding of T and U, and we label every type occurrence in a derived pre-judgement with the path α which corresponds to that occurrence in the unfolding of T or U. In this way, every different definition of a variable X (or t) in the unfolding is associated with a different label α, and we can rename that variable as X_α (or t_α). Hence, we obtain variable uniqueness, and we also preserve, inside t_α, the original name t of the variable it comes from; we will call such t the "face" of the labeled variable t_α. Similar considerations hold for variables X_α.

Since types are ordered binary trees, we represent paths on types by sequences in $\{0,1\}^*$, ranging over by small Greek letters $(\alpha, \beta, \eta, \ldots)$; the empty path is denoted by nil.

In the figure below we have a tree representation of the type $T = \mu X.\forall t \leq \top.\mu Y.t \to X$ where the root is associated with a generic path α:



According to this figure we label the type T as follows:

$$T_l = \mu X^\alpha.\forall t^\alpha \leq \top^{\alpha.0}.\mu Y^{\alpha.1}.(t^{\alpha.1.0} \to X^{\alpha.1.1})$$

which corresponds to the following variable renaming:

$$T_r = \mu X_\alpha.\forall t_\alpha \leq \top.\mu Y_{\alpha.1}.(t_\alpha \to X_\alpha).$$

where each variable has been renamed in accordance with the labeling of the occurrence of its binder.

Finally, we will use the following notation to represent the result of both labeling and renaming a type (we avoid superscripts X^α for typographical reasons).

$$T_{r|l} = \mu X_\alpha.\forall t_\alpha \leq \top_{\alpha.0}.\mu Y_{\alpha.1}.(t_{\alpha|\alpha.1.0} \to X_{\alpha|\alpha.1.1}).$$

Observe that in a type variable occurrence $t_{\alpha|\beta}$, t_α is the variable itself, while β is an occurrence label; the same holds for $X_{\alpha|\beta}$. Also, observe that in $t_{\alpha|\beta}$ the occurrence path β is related to α by $\exists\beta'.\ \beta = \alpha.1.\beta'$. For an occurrence $X_{\alpha|\beta}$ of a recursion variable X_α, the relation $\exists\beta'.\ \beta = \alpha.\beta'$ holds.

Hereafter, for variables $t_{\alpha|\beta}$ and $X_{\alpha|\beta}$, labels α and β will be respectively called *variable label* and *position label*.

For type variables, the position label will be used to prove some good-formation properties of the labeled system we will define (Lemma 4.8 and Corollary 4.9). For recursion variables, the position label will be used to perform renaming in the unfolding rules of the labeled system (Definition 3.11): a recursion variable $X_{\alpha|\beta}$ will be expanded to a type obtained from $\mu X_\alpha.T$ (the type defining the binding for $X_{\alpha|\beta}$) by updating the root α to β and all the internal labels accordingly, without changing variable faces.

Below, we first define a mapping from non-labeled to labeled types and then we define renaming. Hereafter, χ will indicate either a labeled variable t_α or X_α.

To label a type T we have to specify the label β of the root, and a set L that contains the variable label of each free variable in T. Hereafter, such a pair will be denoted as $[L, \beta]$ and called *labeling pair*. Labeling pairs must satisfy a well-formedness condition, which we specify in Definition 3.4.

Definition 3.1 *A labeling set L is a finite set of labeled variables:*

$$L = \{\chi_1, \ldots, \chi_n\}$$

Definition 3.2 *Over $\{0,1\}^*$ we consider the relations $\preceq_p$ (prefix ordering), $\preceq_{rl}$ (right-left ordering), and $\preceq$ (expansion ordering) defined as the least relations such that*

$$(\preceq_p)\ \ \forall\alpha,\beta \in \{0,1\}^* \quad \alpha \preceq_p \alpha.\beta$$

$$(\preceq_{rl})\ \forall\alpha,\beta,\delta \in \{0,1\}^* \ \alpha.1.\delta \preceq_{rl} \alpha.0.\beta$$

$$(\preceq)\ \ \forall\alpha,\beta \in \{0,1\}^* \quad \alpha \preceq \beta \ \Leftrightarrow\ \alpha \preceq_p \beta \text{ or } \alpha \preceq_{rl} \beta$$

As usual, $\prec_p$, $\prec_{rl}$, $\prec$ will denote the irreflexive versions of the corresponding relations.

It is not difficult to prove that $\preceq$ is a reflexive, asymmetric, transitive, and total relation over $\{0,1\}^*$. Indeed, this total order corresponds to a depth-first visit of a binary tree where each node is visited before its descendants and where the right subtrees have priority over the left ones.

Definition 3.3 *For each labeled type T,* $\mathrm{Erase}(T)$ *is the type obtained by erasing each label from variables and $\top_\beta$ that occur in T. Moreover, for each set A of labeled types,* $\mathrm{Erase}^*(A)$ *is the set* $\{\mathrm{Erase}(T) \mid T \in A\}$.

Definition 3.4 (Labeling Pair) *A pair $[L,\beta]$, where $L = \{\chi_1,\ldots,\chi_n\}$, is a labeling pair for $T \in$ R-Types if the following conditions hold, for $i,j = 1\ldots n$:*

1. $\chi_i = t_\alpha \Rightarrow\ \alpha.1 \preceq_p \beta$ (β *is in the scope of* $\forall t_\alpha$)
2. $\chi_i = X_\alpha \Rightarrow\ \alpha.0 \preceq_p \beta\ \vee\ \alpha.1 \preceq_p \beta$ (β *is in the scope of* μX_α)
3. $\mathrm{FV}(T) \subseteq \mathrm{Erase}^*(L)$ (*every free var. in T is defined in L*)
4. $\mathrm{DV}(T) \cap \mathrm{Erase}^*(L) = \emptyset$ (*no defined variable is defined twice*)
5. $i \neq j \Rightarrow\ \mathrm{Erase}(\chi_i) \neq \mathrm{Erase}(\chi_j)$ (*variables in L have different faces*)

We can now define the labeling operator $[L,\beta]\,(T)$.

Definition 3.5 (Labeling) *Let $T \in$ R-Types, if $[L,\beta]$ is a labeling pair for T then $[L,\beta]\,(T)$ is defined by structural induction as follows:*

$$
\begin{aligned}
[L,\beta]\,(\top) &= \top_\beta \\
[L,\beta]\,(t) &= t_{\alpha|\beta} \qquad \textit{where } t_\alpha \in L \\
[L,\beta]\,(X) &= X_{\alpha|\beta} \qquad \textit{where } X_\alpha \in L \\
[L,\beta]\,(\mu X.\forall t \leq T.T') &= \mu X_\beta.\forall t_\beta \leq [L',\beta.0]\,(T).\,[L' \cup \{t_\beta\},\beta.1]\,(T') \\
[L,\beta]\,(\mu X.T \to T') &= \mu X_\beta.\,[L',\beta.0]\,(T) \to [L',\beta.1]\,(T')
\end{aligned}
$$

where $L' = L \cup \{X_\beta\}$.

It is easy to see that the labeling pairs applied to T' and T'' in cases $\forall$ and $\to$ respect Definition 3.4, hence Definition 3.5 is correct.

The result of labeling R-Types is called LR-Types (*Labeled Recursive Types*):

Definition 3.6 (LR-Types)

$$
\begin{aligned}
\textit{LR-Types} = \{T \mid\ & \textit{exists } T' \in \textit{R-Types and a labeling pair } [L,\beta] \textit{ for } T' \\
& \textit{such that } T = [L,\beta]\,(T')\}
\end{aligned}
$$

Property 3.7 (Face determinism) *For any type $T \in$ LR-Types, if χ and χ' both occur in T then* $\mathrm{Erase}(\chi) = \mathrm{Erase}(\chi') \Rightarrow \chi = \chi'$.

Proof. By induction on the structure of T and by using $\mathrm{Erase}(T) \in$ R-Types and uniqueness of variable names in R-Types.□

Definition 3.8 *For each type $T \in$ LR-Types we define* $\mathrm{DV}(T)$ *and* $\mathrm{FV}(T)$*, as follows:*

$$
\begin{array}{ll}
\mathrm{DV}(\top_\beta) & = \emptyset \\
\mathrm{DV}(t_{\alpha|\beta}) & = \emptyset \\
\mathrm{DV}(X_{\alpha|\beta}) & = \emptyset \\
\mathrm{DV}(\mu X_\alpha.\forall t_\alpha \leq T'.U') & = \{X_\alpha, t_\alpha\} \cup \mathrm{DV}(T') \cup \mathrm{DV}(U') \\
\mathrm{DV}(\mu X_\alpha.T' \to U') & = \{X_\alpha\} \cup \mathrm{DV}(T') \cup \mathrm{DV}(U')
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{FV}(\top_\beta) & = \emptyset \\
\mathrm{FV}(t_{\alpha|\beta}) & = \{t_\alpha\} \\
\mathrm{FV}(X_{\alpha|\beta}) & = \{X_\alpha\} \\
\mathrm{FV}(\mu X_\alpha.T' \to U') & = (\mathrm{FV}(T') \cup \mathrm{FV}(U')) - \{X_\alpha\} \\
\mathrm{FV}(\mu X_\alpha.\forall t_\alpha \leq T'.U') & = (\mathrm{FV}(T') \cup (\mathrm{FV}(U') - \{t_\alpha\})) - \{X_\alpha\}
\end{array}
$$

The labeling operator allows an R-Type to be transformed into an LR-Type. We will also use a relabeling operator that takes an LR-Type T and a label β and updates the root label of T to β, and every other bound variable accordingly; this operator does not modify the variable label of the free variables, but only their position labels.

We now define when a pair $\langle L, \beta \rangle$ is a *relabeling pair* for T.

Notation 3.9 *If $T \in$ LR-Types,* $\mathrm{Root}(T)$ *is the root label of T. Formally, if*

$$T = \top_\beta \mid t_{\alpha|\beta} \mid X_{\alpha|\beta} \mid \mu X_\beta.\forall t_\beta \leq T'.U' \mid \mu X_\beta.T' \to U'$$

then $\mathrm{Root}(T) = \beta$.

Definition 3.10 *The pair $\langle L, \beta \rangle$ is a relabeling pair for $T \in$ LR-Types if and only if the following conditions hold:*

1. $\mathrm{Root}(T) \preceq_p \beta$
2. $[L, \beta]$ *is a labeling pair for* $\mathrm{Erase}(T)$

Relabeling of LR-Types is defined as follows.

Definition 3.11 *Let $T \in$ LR-Types, if $\langle L, \beta \rangle$ is a relabeling pair for T, then $\langle L, \beta \rangle (T)$ is defined as follows:*

$$\langle L, \beta \rangle (T) = [L, \beta] (\text{Erase}(T))$$

Lemma 3.12 *If $T \in$ LR-Types and $\langle L, \beta \rangle$ is a relabeling pair for T, then $\langle L, \beta \rangle (T) \in$ LR-Types.*

Proof. By definition of LR-Types and by observing that if $\langle L, \beta \rangle$ is a relabeling pair for T, then $[L, \beta]$ is a labeling pair for Erase(T).□

As already stated, relabeling is used when an occurrence of a recursion variable $X_{\alpha|\beta}$ is substituted with its body $\mu X_\alpha.T$, during subtype checking. In this case, we will guarantee the uniqueness of variables by expanding $X_{\alpha|\beta}$ to $\langle \text{FV}(\mu X_\alpha.T), \beta \rangle (\mu X_\alpha.T)$.

Notation 3.13 *Hereafter, $\langle \text{FV}(\mu X_\alpha.T), \beta \rangle (\mu X_\alpha.T)$ will be abbreviated with $(\mu X_\alpha.T) \uparrow \beta$.*

Over LR-Types we consider the following equivalence relation.

Definition 3.14 (Similarity) *For $T, U \in$ LR-Types, $T \simeq U \Leftrightarrow_{def}$* Erase($T$) = Erase($U$).

The relation $T \simeq U$ (T *similar to* U) will be used to define the stop condition of our subtype checking algorithm (Sections 4.2-4.3).

Definition 3.15 *For each $T \in$ LR-Types and $A, B \subseteq$ LR-Types, we say that $T \in^{\simeq} A$ (T is in A modulo $\simeq$ relation) if* Erase(T) $\in$ Erase*(A), *$A \subseteq^{\simeq} B$ if* Erase*(A) $\subseteq$ Erase*(B), *while $A \cap^{\simeq} B$ denotes the set* Erase*(A) $\cap$ Erase*(B).

We now give some properties concerning types in LR-Types, which will be used in the rest of the paper to prove soundness of the subtyping algorithm we are going to present. First we need the following definition.

Definition 3.16 *For each $T \in$ LR-Types,* SE(T) *denotes the set of all subexpressions of T:*

$$\begin{aligned}
&\text{SE}(\top_\beta) &&= \{\top_\beta\} \\
&\text{SE}(t_{\alpha|\beta}) &&= \{t_{\alpha|\beta}\} \\
&\text{SE}(X_{\alpha|\beta}) &&= \{X_{\alpha|\beta}\} \\
&\text{SE}(\mu X_\alpha.\forall t_\alpha \leq T.U) &&= \{\mu X_\alpha.\forall t_\alpha \leq T.U\} \cup \text{SE}(T) \cup \text{SE}(U) \\
&\text{SE}(\mu X_\alpha.T \to U) &&= \{\mu X_\alpha.T \to U\} \cup \text{SE}(T) \cup \text{SE}(U)
\end{aligned}$$

Observe that for each $T \in$ LR-Types, SE(T) is a finite set.

Lemma 3.17 *If $T \in$ LR-Types then* $\mathrm{SE}(T) \subseteq$ *LR-Types.*

Lemma 3.18 *If $T \in$ LR-Types, then for each $T', U' \in \mathrm{SE}(T)$:*

$$\mathrm{Root}(T') \preceq_p \mathrm{Root}(U') \;\Leftrightarrow\; U' \in \mathrm{SE}(T')$$

Proof. By structural induction on T and by definition of labeling (Definition 3.5).□

Lemma 3.19 *If $T \in$ LR-Types, then for any t_β and X_β:*

$$t_\beta \in \mathrm{FV}(T) \Rightarrow \beta \prec_p \mathrm{Root}(T) \qquad X_\beta \in \mathrm{FV}(T) \Rightarrow \beta \prec_p \mathrm{Root}(T) \qquad (1)$$

$$t_\beta \in \mathrm{DV}(T) \Rightarrow \mathrm{Root}(T) \preceq_p \beta \qquad X_\beta \in \mathrm{DV}(T) \Rightarrow \mathrm{Root}(T) \preceq_p \beta \qquad (2)$$

Proof. By structural induction on T and by definition of labeling (Definition 3.5).□

Lemma 3.20 *If $T \in$ LR-Types, then for each $T', U' \in \mathrm{SE}(T)$:*

$$\mathrm{FV}(T') \cap \mathrm{DV}(U') \neq \emptyset \;\Rightarrow\; T' \in \mathrm{SE}(U')$$

Proof. Let $\chi \in FV(T') \cap \mathrm{DV}(U')$, where $\chi = u_\beta$ or $\chi = Y_\beta$; by Lemma 3.19,

$$\mathrm{Root}(U') \preceq_p \beta \prec_p \mathrm{Root}(T');$$

the claim follows by Lemma 3.18.□

Lemma 3.21 *If $T \in$ LR-Types, then for each $T', U' \in \mathrm{SE}(T)$:*

$$\mathrm{FV}(T') \cap^{\simeq} \mathrm{DV}(U') \neq \emptyset \;\Rightarrow\; T' \in \mathrm{SE}(U')$$

Proof. By hypothesis we have two variables χ and χ' with the same face such that $\chi \in \mathrm{FV}(T')$ and $\chi' \in \mathrm{DV}(T')$ and both occur in T. By Property 3.7 (face determinism), this implies that $\chi = \chi'$, hence $\mathrm{FV}(T') \cap \mathrm{DV}(U') \neq \emptyset$, therefore Lemma 3.20 can be applied.□

The lemma just proved can be generalized to the following corollary, where $\in$ is substituted by $\in^{\simeq}$.

Corollary 3.22 *If $T \in$ LR-Types, then for each $T', U' \in^{\simeq} \mathrm{SE}(T)$:*

$$\mathrm{FV}(T') \cap^{\simeq} \mathrm{DV}(U') \neq \emptyset \;\Rightarrow\; T' \in^{\simeq} \mathrm{SE}(U')$$

Proof. By hypothesis there exist $T'', U'' \in \mathrm{SE}(T)$ such that $T'' \simeq T'$ and $U'' \simeq U'$ with $\mathrm{FV}(T'') \cap^{\simeq} \mathrm{DV}(U'') \neq \emptyset$. Hence, by Lemma 3.21, $T'' \in \mathrm{SE}(U'')$, which implies $T' \in^{\simeq} \mathrm{SE}(U')$ by $T'' \simeq T'$ and $U'' \simeq U'$.□

We are now ready to provide a precise definition of labeled recursive kernel Fun. This system is strictly related to the one defined in Section 2. One novel feature is variable labeling and relabeling, which is used to perform renaming in the unfolding rules. Moreover, in this variant, when a comparison $X_{\alpha|\beta} \leq U$ or $T \leq Y_{\nu|\eta}$ is met during subtype checking, this information is saved in the bi-environment. This is only done for uniformity with the algorithmic version of the next section, where this information will be used to stop the subtype-checking process. In this abstract version, this information is not used.

Hereafter, $T_{X,\alpha}$ will denote a type $\mu X_\alpha.\forall t_\alpha \leq T'.U'$ or $\mu X_\alpha.T' \to U'$, while $\diamond$ will range over $\leq$ and $\geq$; if $\diamond$ is $\leq$ ($\geq$), then $\diamond^{-1}$ is $\geq$ (resp., $\leq$).

The syntax for types, pre-judgements and judgements is as follows:

Types	T, U	$::= \top_\beta \mid t_{\alpha\|\beta} \mid X_{\alpha\|\beta} \mid \mu X_\alpha.\forall t_\alpha \leq T.U$
		$\mid \mu X_\alpha.T \to U$
Pre-Judgements	P	$::= \Gamma \rhd_\ell \text{ Env} \mid \Gamma \rhd_\ell T \text{ Type} \mid \Pi \rhd_\ell T \leq U$
Judgements	J	$::= \Gamma \vdash_\ell \text{ Env} \mid \Gamma \vdash_\ell T \text{ Type} \mid \Pi \vdash_\ell T \leq U$
Bi-Environments	Π	$::= () \mid \Pi, (t_\alpha, u_\beta) \leq (T, U)$
		$\mid \Pi, (X_\alpha = T_{X,\alpha}, Y_\delta = U_{Y,\delta})$
		$\mid \Pi, X_{\alpha\|\beta} \diamond T \mid \Pi, T \diamond X_{\alpha\|\beta}$
Environments	Γ	$::= () \mid \Gamma, t_\alpha \leq T \mid \Gamma, X_\alpha = T_{X,\alpha} \mid \Gamma, T$

The Def, Left, Right and Swap operations are now defined as in the following table; observe that now the Swap operation, used in rule ($\to_\leq$), also swaps positions of types in assumptions $X_{\alpha|\beta} \leq T$ and $T \leq X_{\alpha|\beta}$ without changing their meaning, thus yielding $T \geq X_{\alpha|\beta}$ and $X_{\alpha|\beta} \geq T$, respectively.

	$(t_\alpha, u_\beta) \leq (T, U)$	$(X_\alpha = T, Y_\delta = U)$	$T \diamond U$
Def	(t_α, u_β)	(X_α, Y_δ)	
Swap	$(u_\beta, t_\alpha) \leq (U, T)$	$(Y_\delta = U, X_\alpha = T)$	$U \diamond^{-1} T$
Left	$t_\alpha \leq T$	$X_\alpha = T$	T
Right	$u_\beta \leq U$	$Y_\delta = U$	U

We now present the rules that define our subtype relation over recursive types; we will call this set of rules $\Re^\infty$. These rules will be interpreted coinductively (Definition 3.25).

Hereafter, $u_{\nu|-}$ indicates that the position label is an arbitrary label; likewise for $\top_-$.

3.2.0.3 Subtyping rules

$$\frac{\Pi \vdash_\ell T, \top \text{ Type}}{\Pi \vdash_\ell T \leq \top_\beta} \quad (\top_\leq) \qquad \frac{(t_\alpha, u_\nu) \in \text{Def}(\Pi) \;\; \Pi \vdash_\ell t_{\alpha|\beta}, u_{\nu|\eta} \text{ Type}}{\Pi \vdash_\ell t_{\alpha|\beta} \leq u_{\nu|\eta}} \quad (\text{Id}_\leq)$$

$$\frac{\begin{array}{l}(t_\alpha, u_\nu) \leq (T', U') \in \Pi \\ \textit{for all } X_{\theta|\delta}.\ (U \neq X_{\theta|\delta}) \quad U \neq u_{\nu|-} \quad U \neq \top_- \quad \Pi \vdash_\ell T' \leq U\end{array}}{\Pi \vdash_\ell t_{\alpha|\beta} \leq U} \quad (\text{VarTrans}_\leq)$$

$$\frac{\begin{array}{l}\Pi' = \Pi, (X_\alpha = \mu X_\alpha.T \to U, \;\; Y_\nu = \mu Y_\nu.T' \to U') \\ \text{Swap}(\Pi') \vdash_\ell T' \;\leq\; T \quad \Pi' \vdash_\ell U \;\leq\; U'\end{array}}{\Pi \vdash_\ell \mu X_\alpha.T \to U \;\leq\; \mu Y_\nu.T' \to U'} \quad (\to_\leq)$$

$$\frac{\begin{array}{l}\Pi' = \Pi, (X_\alpha = \mu X_\alpha.\forall t_\alpha \leq T.U, \;\; Y_\nu = \mu Y_\nu.\forall u_\nu \leq T'.U') \\ \Pi' \vdash_\ell T \triangleq T' \quad \Pi', (t_\alpha, u_\nu) \leq (T, T') \vdash_\ell U \leq U'\end{array}}{\Pi \vdash_\ell \mu X_\alpha.\forall t_\alpha \leq T.U \;\leq\; \mu Y_\nu.\forall u_\nu \leq T'.U'} \quad (\forall_\leq)$$

$$\frac{\begin{array}{l}X_\alpha = T \in \text{Left}(\Pi) \\ \Pi, X_{\alpha|\beta} \leq U \vdash_\ell T \uparrow \beta \;\leq\; U\end{array}}{\Pi \vdash_\ell X_{\alpha|\beta} \;\leq\; U} \quad (\text{LUnf}_\leq)$$

$$\frac{\begin{array}{l}\textit{for all } X_{\theta|\delta}.\ (T \neq X_{\theta|\delta}) \quad Y_\nu = U \in \text{Right}(\Pi) \\ \Pi, T \leq Y_{\nu|\eta} \vdash_\ell T \;\leq\; U \uparrow \eta\end{array}}{\Pi \vdash_\ell T \;\leq\; Y_{\nu|\eta}} \quad (\text{RUnf}_\leq)$$

Observe that the unfolding rules now use the relabeling operation to rename the unfolded types. We will see (Corollary 4.9) that this renaming is sufficient to avoid name clashes and to preserve well-formedness. Moreover, we will give sufficient and non restrictive conditions that will make this renaming always applicable (Definition 3.23). Observe that no renaming is performed by rule ($\text{VarTrans}_\leq$).

If a pre-judgement P is reduced to P' by one backward application of an $\Re^\infty$ rule called (R), then we indicate this fact with

$$P \xrightarrow{(R)}_\infty P'$$

while $P \twoheadrightarrow_\infty P'$ means that either $P = P'$ or P is reduced to P' by one or more backward applications of $\Re^\infty$ rules.

As for the non-labeled system, rule application is made deterministic by inequality conditions expressed in rules (VarTrans$_\leq$) and (RUnf$_\leq$).

The restriction to closed judgement we made in Section 2 is formally extended to the labeled system by definitions 3.23 and 3.24 .

Definition 3.23 *We call Start-J the set of pre-judgements* $() \rhd_\ell T \leq U$ *where* T, U *are closed LR-Types and:*

$$\mathrm{DV}(T) \cap^{\simeq} \mathrm{DV}(U) = \emptyset$$

The non restrictive condition of uniqueness of type faces will be helpful later in simplifying the statement of some properties (see Lemma 5.19).

Definition 3.24 *We define Start-*J^∞ *as the set of all subtyping pre-judgements that we obtain by reducing a pre-judgement in Start-J by backward applications of* $\Re^\infty$ *rules. Formally:*

$$\textit{Start-}J^\infty = \{P' : \exists\, P \in \textit{Start-J s.t. } P \twoheadrightarrow_\infty P'\}$$

The Start-J$^\infty$ pre-judgements satisfy some invariants that we prove in Section 4.4; in particular, they are well-formed.

We now give our definition of inclusion between labeled recursive types; a *failure pre-judgement* is a pre-judgement that is not equal to the conclusion of any ground instance of any rule. We indicate with $\Pi \vdash_\ell^\infty T \leq U$ the fact that $\Pi \vdash_\ell T \leq U$ holds with respect to $\Re^\infty$ rules.

Definition 3.25 *For each pre-judgement* $\Pi \rhd_\ell T \leq U$ *in Start-*J^∞:

$$\Pi \vdash_\ell^\infty T \leq U \Leftrightarrow \nexists \textit{ a failure pre-judgement } \Pi' \rhd_\ell T' \leq U'$$
$$\textit{s.t. } \Pi \rhd_\ell T \leq U \twoheadrightarrow_\infty \Pi' \rhd_\ell T' \leq U'$$

Since every judgement has only one rule that may be used to prove it (rule determinism), the previous definition can be restated as: $\Pi \vdash_\ell^\infty T \leq U$ if either a finite or an infinite proof tree exists for it.

Good formation rules for the labeled system are the following. (Γ TypeForm) has been added to check that types coming from assumptions $T \diamond U$ are well formed.

3.2.0.4 Labeled good formation rules

$$\frac{}{() \vdash_\ell \text{Env}} \ (\Gamma \text{ EmptyForm}) \qquad \frac{t_\alpha \notin \text{Def}(\Gamma) \quad \Gamma \vdash_\ell T \text{ Type}}{\Gamma, t_\alpha \leq T \vdash_\ell \text{Env}} \ (\Gamma \text{ BoundForm})$$

$$\frac{\Gamma \vdash_\ell T \text{ Type}}{\Gamma, T \vdash_\ell \text{Env}} \ (\Gamma \text{ TypeForm})$$

$$\frac{X_\alpha \notin \text{Def}(\Gamma) \quad \Gamma \vdash_\ell T \text{ Type}}{\Gamma, X_\alpha = T \vdash_\ell \text{Env}} \ (\Gamma \text{ EqForm})$$

$$\frac{t_\alpha \leq T \in \Gamma \quad \Gamma \vdash_\ell T \text{ Type}}{\Gamma \vdash_\ell t_{\alpha|\beta} \text{ Type}} \ (\text{T-VarForm})$$

$$\frac{\Gamma \vdash_\ell \text{Env}}{\Gamma \vdash_\ell \top_\beta \text{ Type}} \ (\top \text{ Form}) \qquad \frac{X_\alpha \in \text{Def}(\Gamma) \quad \Gamma \vdash_\ell \text{Env}}{\Gamma \vdash_\ell X_{\alpha|\beta} \text{ Type}} \ (\text{R-VarForm})$$

$$\frac{\Gamma, X_\alpha = \top_\alpha \vdash_\ell T \text{ Type} \quad \Gamma, X_\alpha = \top_\alpha \vdash_\ell U \text{ Type}}{\Gamma \vdash_\ell \mu X_\alpha.T \to U \text{ Type}} \ (\to \text{ Form})$$

$$\frac{\Gamma, X_\alpha = \top_\alpha, t_\alpha \leq T \vdash_\ell U \text{ Type}}{\Gamma \vdash_\ell \mu X_\alpha.\forall t_\alpha \leq T.U \text{ Type}} \ (\forall \text{ Form})$$

Note that in the premise of (T-VarForm) and (R-VarForm) we ignore the position label; as stated before, this label is only needed for renaming in the subtyping rules.

The labeled system we have presented here is equivalent to the system defined in Section 2, which means that the same closed judgements hold modulo labeling. To prove equivalence we need the following definition, where we define equality up to variable names between R-Types and LR-Types; in the same definition we extend this equality to non-labeled bi-environments and labeled bi-environments.

Definition 3.26 *For any injective function f from non-labeled variables to labeled variables, we define the binary relation $\doteq_f \subseteq$ R-Types$\times$LR-Types as follows*

$$\begin{aligned}
\top &\doteq_f \top_\alpha \\
t &\doteq_f t'_{\alpha|\beta} && \Leftrightarrow f(t) = t'_\alpha \\
X &\doteq_f X'_{\alpha|\beta} && \Leftrightarrow f(X) = X'_\alpha \\
\mu X.T \to U &\doteq_f \mu X'_\alpha.T' \to U' && \Leftrightarrow T \doteq_f T',\ U \doteq_f U',\ f(X) = X'_\alpha \\
\mu X.\forall t \leq T.U &\doteq_f \mu X'_\alpha.\forall t'_\alpha \leq T'.U' && \Leftrightarrow T \doteq_f T',\ U \doteq_f U',\ f(X) = X'_\alpha, \\
& && \quad f(t) = t'_\alpha
\end{aligned}$$

We extend $\doteq_f$ to bi-environments and pre-judgements as follows

$$() \doteq_f ()$$

$$\Pi, (t,u) \leq (T,U) \doteq_f \Pi', (f(t), f(u)) \leq (T', U') \Leftrightarrow \Pi \doteq_f \Pi',\ T \doteq_f T',\ U \doteq_f U'$$

$$\Pi, (X = T, Y = U) \doteq_f \Pi', (f(X) = T', f(Y) = U') \Leftrightarrow \Pi \doteq_f \Pi',\ T \doteq_f T',\ U \doteq_f U'$$

$$\Pi \doteq_f \Pi', T' \diamond U' \Leftrightarrow \Pi \doteq_f \Pi',\ \Pi' \vdash_\ell T', U'\ \mathit{Type}$$

$$\Pi \rhd T \leq U \doteq_f \Pi' \rhd_\ell T' \leq U' \Leftrightarrow \Pi \doteq_f \Pi',\ T \doteq_f T',\ U \doteq_f U'$$

Now we can prove equivalence.

Theorem 3.27 *For each injective function f and each pre-judgement $\Pi \rhd T \leq U$ in recursive kernel Fun,*

$$\Pi \rhd T \leq U \ \doteq_f\ \Pi' \rhd_\ell T' \leq U' \Rightarrow (\Pi \vdash T \leq U \Leftrightarrow \Pi' \vdash_\ell^\infty T' \leq U')$$

Proof. Observe that, for any f and $\Pi \rhd T \leq U$:

$$\Pi \rhd T \leq U \ \doteq_f\ \Pi' \rhd_\ell T' \leq U' \ \wedge\ \Pi \rhd T \leq U \xrightarrow{(R)} \Pi_1 \rhd T_1 \leq U_1$$
$$\Rightarrow\ \exists f', \Pi_1', T_1', U_1'.\ \Pi_1 \rhd T_1 \leq U_1 \ \doteq_{f'}\ \Pi_1' \rhd_\ell T_1' \leq U_1' \wedge \Pi' \rhd_\ell T' \leq U' \xrightarrow{(R)}_\infty \Pi_1' \rhd_\ell T_1' \leq U_1'$$

$$\Pi \rhd T \leq U \ \doteq_f\ \Pi' \rhd_\ell T' \leq U' \ \wedge\ \Pi' \rhd_\ell T' \leq U' \xrightarrow{(R)}_\infty \Pi_1' \rhd_\ell T_1' \leq U_1'$$
$$\Rightarrow\ \exists f', \Pi_1, T_1, U_1.\ \Pi_1 \rhd T_1 \leq U_1 \ \doteq_{f'}\ \Pi_1' \rhd_\ell T_1' \leq U_1' \wedge \Pi \rhd T \leq U \xrightarrow{(R)} \Pi_1 \rhd T_1 \leq U_1$$

and observe that $\Pi \rhd T \leq U \ \doteq_f\ \Pi' \rhd_\ell T' \leq U'$ implies that one judgement is a failure judgement if and only if the other one is. Then conclude that $\Pi' \rhd_\ell T' \leq U'$ is reduced to a failure judgement if and only if $\Pi \rhd T \leq U$ is reduced to a failure judgement as well.□

Corollary 3.28 *For each closed pre-judgement $() \rhd_\ell T \leq U$,*

$$() \vdash_\ell^\infty T \leq U \Leftrightarrow () \vdash \mathrm{Erase}(T) \leq \mathrm{Erase}(U)$$

Corollary 3.29 *For each closed pre-judgement* $() \rhd T \leq U$ *in recursive kernel Fun,*

$$() \vdash T \leq U \Leftrightarrow () \vdash_{\ell}^{\infty} [\{\}, \alpha]\,(T) \leq [\{\}, \alpha]\,(U)$$

4 A Subtyping Algorithm

In this section we provide an alternative set of rules for defining the subtype relation over labeled recursive types. This set of rules defines an algorithm since at most one rule can be selected for any pre-judgement, and the backward application of these rules always terminates.

The rules presented in the previous section record the pairs $X_{\alpha|\beta} \leq U$ or $T \leq Y_{\nu|\eta}$ in the bi-environments. Thus, following [AC93], we can use this information to stop backward rule application when such a pair of types is met for the second time. Due to renaming, we cannot expect exactly the same pair to be met twice. Hence the most natural idea is to stop when we meet a pair that matches an already met pair modulo α-renaming.

This algorithm is very inefficient because of the high cost of α-equivalence comparison, but, prior to this study, it was regarded as the best guess for a correct and complete algorithm. We will show in Section 4.1 that this is not the case, and we consider this as an important, though negative, result. This algorithm is correct, but it is not complete since there exist provable judgements that, during subtype checking, produce infinitely many pairs that, though in some sense "similar", always fail to be α-equivalent to a previously met pair.

The counterexample in Section 4.1 suggests that, in order to define a complete subtyping algorithm, one may look for a more "syntactical" equivalence relation to be used in the stop condition in place of α-equivalence. Similarity (i.e. erasure equality $\simeq$) is the first candidate for this task. Unfortunately, it is too weak. The resulting algorithm is complete, i.e. it always terminates, but is not correct, as shown in Section 4.2.

However, the algorithm becomes complete and correct if we use similarity but, instead of stopping the second time we meet a pair, we wait until the same pair is met, modulo similarity, for the third time, as formalized in Section 4.3.

We show here that the algorithm that stops when it meets a pair that is α-equivalent to an already met pair is incomplete, i.e. it diverges on a provable judgement.

To define the α-equivalence based algorithm we consider the following end-rule to be added to $\Re^{\infty}$; the resulting set of rules will be called $\Re^{alg\text{-}\alpha}$; α-equivalence "$\simeq_\alpha$" is defined as usual: free variables must be equal, while bound variables can be freely renamed.

$$\frac{T' \leq U' \in \Pi \quad T \simeq_\alpha T' \;\; U \simeq_\alpha U' \;\; \Pi \vdash_\ell T, U \text{ Type}}{\Pi \vdash_\ell T \leq U} \quad (\text{End}^{\alpha}_{\leq})$$

Our complete algorithm (see Section 4.3) records a pair $T \leq U$ only when the unfolding rule is applied, which gives the end-rule fewer possibilities of being applicable, but is still enough to make it complete. We show here that the incompleteness of the α-based algorithm is not a consequence of this choice, by showing the divergence of an algorithm that records *every* met pair.

To preserve determinism, we assume that the $(\text{End}^{\alpha}_{\leq})$ rule takes priority over all the other ones except for the termination rules $(\text{Id}_{\leq})$ and $(\top_{\leq})$. No subtyping is lost since $(\text{End}^{\alpha}_{\leq})$, $(\text{Id}_{\leq})$ and $(\top_{\leq})$ are all termination rules.

The two relations $\xrightarrow{(R)}_{alg\text{-}\alpha}$, $\twoheadrightarrow_{alg\text{-}\alpha}$ are defined similarly to $\xrightarrow{(R)}_{\infty}$ and $\twoheadrightarrow_{\infty}$. In the $\Re^{alg\text{-}\alpha}$ system, a prejudgement $() \rhd_\ell T \leq U$ is provable if and only if the $\Re^{alg\text{-}\alpha}$ backward reduction process starting from $() \rhd_\ell T \leq U$ terminates without failing.

The $() \rhd_\ell T \leq U$ system is not complete: the stop condition expressed by the $(\text{End}^{\alpha}_{\leq})$ rule does not guarantee termination, not even for $\Re^{\infty}$ provable pre-judgements. We prove this fact by showing a $\Re^{\infty}$ provable pre-judgement that makes the algorithm diverge.

To this end, we first fix some conventions that we will use in the proof. Hereafter we assume that:

- $\forall t_\alpha.T$ denotes the type body $\forall t_\alpha \leq \top.T$ where t_α may occur free in T;
- $\forall.T$ denotes a type body $\forall t_\alpha \leq \top.T$ where t_α does not occur free in T;
- $\mu.T$ denotes a type $\mu X_\alpha.T$ where X_α does not occur free in the body T.

Moreover, for simplicity, in variables $t_{\alpha|\beta}$ and $X_{\alpha|\beta}$ we omit position labels β and only retain variable labels α. Each time a variable is renamed, new labels are simply indicated with new names (e.g. t_α is renamed in t_ρ with $\alpha \neq \rho$).

To simplify things, in the diverging pre-judgement we will use a type $\bot$ that is the subtype of every type, associated with the following termination rule:

$$\frac{\Pi \vdash_\ell \bot_\beta, T \text{ Type}}{\Pi \vdash_\ell \bot_\beta \leq T} \quad (\bot_\leq)$$

We will also use pair types $\mu X_\alpha.T \times U$ with the usual covariant rule, corresponding to the following pair of reduction rules, where Π'' is defined as in rule $(\rightarrow_\leq)$ and where it is explicitly indicated if the reduction is to the left or right premise.

$$\Pi \rhd_\ell \mu X_\alpha.T \times U \leq \mu Y_\nu.T' \times U' \xrightarrow[alg\text{-}\alpha]{(\times_\leq)^l} \Pi'' \rhd_\ell T \leq T'$$
$$\Pi \rhd_\ell \mu X_\alpha.T \times U \leq \mu Y_\nu.T' \times U' \xrightarrow[alg\text{-}\alpha]{(\times_\leq)^r} \Pi'' \rhd_\ell U \leq U'$$

For product types we will also use the following notation:

- $T \times U$ denotes a product type $\mu X_\alpha.T \times U$ where X_α does not occur free in $T \times U$.

Pair and bottom types make our counterexample much more readable, and we can encode both of them in system kernel Fun, provided that $\bot$ is not used as the bound of a type variable (see Appendix C).

We can now present the diverging pre-judgement:

$$() \rhd_\ell \overline{T} \leq \overline{U}$$

where

$$\overline{T} = \mu.\forall.\mu X_\alpha.\forall t_\alpha.(\bot \times \mu Z_\gamma.\forall.\mu.\forall.((\bot \times t_\alpha \times X_\alpha) \times Z_\gamma))$$
$$\overline{U} = \mu Y_\theta.\forall u_\theta.\mu K_\tau.\forall.((u_\theta \times \top \times K_\tau) \times Y_\theta)$$

Since these types are quite complex, we will split them, and will name their parts as follows. Each name is parametrized with respect to the variables that appear free in the corresponding type.

$$\overline{T} = \mu.\forall.\mu X_\alpha.\forall t_\alpha.\ (A_{X_\alpha,t_\alpha}) \quad A_{X_\alpha,t_\alpha} = (\bot \times \mu Z_\gamma.\forall.\mu.\forall.\ B_{X_\alpha,t_\alpha,Z_\gamma})$$
$$B_{X_\alpha,t_\alpha,Z_\gamma} = (C_{X_\alpha,t_\alpha} \times Z_\gamma) \qquad C_{X_\alpha,t_\alpha} = (\bot \times t_\alpha \times X_\alpha)$$

$$\overline{U} = \mu Y_\theta.\forall u_\theta.\mu K_\tau.\forall.\ (D_{Y_\theta,u_\theta,K_\tau}) \quad D_{Y_\theta,u_\theta,K_\tau} = (E_{u_\theta,K_\tau} \times Y_\theta)$$
$$E_{u_\theta,K_\tau} = (u_\theta \times \top \times K_\tau)$$

Both types $\overline{T}$ and $\overline{U}$ are characterized by the nesting pattern $\mu X.\, \forall t.\, \mu Z.\, T_{X,t,Z}$: a type variable is defined between two recursion variables, and the three of them appear in the internal scope. As we will see, this binder alternation is crucial in order to make both systems, $\Re^{\infty}$ and $\Re^{alg\text{-}\alpha}$, diverge, and this divergence makes the pre-judgement provable in system $\Re^{\infty}$ and not in system $\Re^{alg\text{-}\alpha}$.

The behavior of the type-checking algorithm is better explained through a figure. Figure 1 illustrates how $\overline{T}$ and $\overline{U}$ are compared, and how they are unfolded during the comparison. It shows how the repeated unfolding of the two trees generates two long skeletons that have exactly the same structure, along which the comparison goes on forever. Indeed, after six levels, similar types are found, and similar pairs of types are subject to the application of the same rule, producing another similar pair, and so on. There is only one situation where similar comparisons may be reduced differently by the subtyping rules, which is variable-to-variable comparison: $u_\alpha \leq t_\beta$ is similar to $u_\gamma \leq t_\delta$, but rule ($\mathrm{Id}_{\leq}$) may be applicable to the first while rule ($\mathrm{VarTrans}_{\leq}$) is applicable to the other (or vice versa), as exemplified in the next section. But no variable-to-variable comparison is performed here, since variables are only compared with $\bot$ or with $\top$. And no pair of compared types is ever going to be α-similar to one that has been met before. Consider for example the two similar pairs met at the second and eighth levels of the tree (we use superscripts like $^{\alpha}$, $^{\rho}$, etc., to distinguish among α-equivalent types that have no free variables to write in the subscript):

$$L^{\alpha} \leq R_{Y_\theta,u_\theta} \text{ i.}e.\ \mu X_\alpha.\forall t_\alpha.\ (A_{X_\alpha,t_\alpha}) \leq \mu K_\tau.\forall.\ (D_{Y_\theta,u_\theta,K_\tau})$$
$$L^{\rho} \leq R_{Y_\beta,u_\beta} \text{ i.}e.\ \mu X_\rho.\forall t_\rho.\ (A_{X_\rho,t_\rho}) \ \leq \mu K_\phi.\forall.\ (D_{Y_\beta,u_\beta,K_\phi})$$

The comparison does not stop here, since R_{Y_θ,u_θ} is not α-equivalent to R_{Y_β,u_β}, because of the different sets of free variables. After a couple of steps, in the tenth line, the right-hand-side R'^{ν} gets rid of the free variables, and becomes α-equivalent to R'^{β} met at the fourth level. But, at this point, the left-hand-side L'_{X_ρ,t_ρ} has acquired two free variables that make it non equivalent to L'_{X_α,t_α}. These free variables will stay until the next repetition of the $L^{-} \leq R_{Y_{\_},u_{\_}}$ pair.

We now give a more formal and detailed account of this divergence. We first present an initial part of the infinite $\Re^{alg\text{-}\alpha}$ reduction chain that originates from $\overline{T} \leq \overline{U}$.

In each reduction step we only write the last element added to the previous bi-environment, and we assume that the current comparison is saved in the bi-environment, without writing this down explicitly. Finally, we omit proof branches concerning well-formedness (as already mentioned, in Lemma 4.14

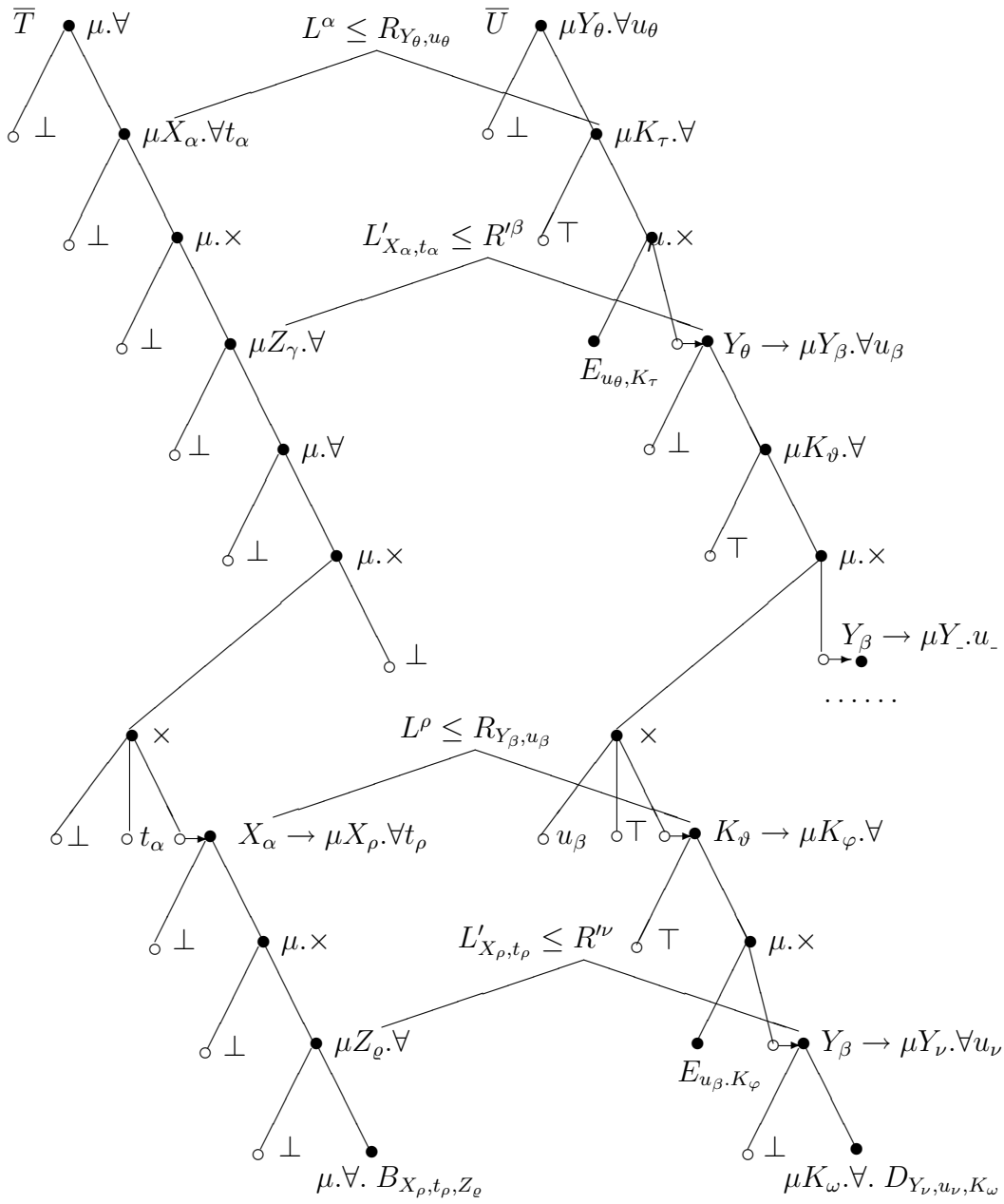


Fig. 1. How $\overline{T}$ and $\overline{U}$ are expanded and traversed

we prove that, for Start-J$^{\infty}$ pre-judgements, well-formedness is always guaranteed).

$$(1) \qquad () \rhd_\ell \mu.\forall.\mu X_\alpha.\forall t_\alpha.\ (A_{X_\alpha,t_\alpha}) \ \leq\ \mu Y_\theta.\forall u_\theta.\mu K_\tau.\forall.\ (D_{Y_\theta,u_\theta,K_\tau})$$

$$(2) \ \xrightarrow{(\forall\leq)}_{\textit{alg-}\alpha} \ \ldots\ (-{=}\overline{T}\,,\ Y_\theta{=}\overline{U}),\ (-, u_\theta) \leq (\top,\top) \rhd_\ell$$

$$\mu X_\alpha.\forall t_\alpha.(A_{X_\alpha,t_\alpha}) \leq \mu K_\tau.\forall.(D_{Y_\theta,u_\theta,K_\tau})$$

(3) $\xrightarrow{(\forall\leq)}_{alg\text{-}\alpha}$ $\ldots (X_\alpha = \mu X_\alpha.\forall t_\alpha.(A_{X_\alpha,t_\alpha}),\ \ K_\tau = \mu K_\tau.\forall.\ (D_{Y_\theta,u_\theta,K_\tau})),$
$(t_\alpha, -) \leq (\top,\top) \rhd_\ell$
$(\bot \times \mu Z_\gamma.\forall.\mu.\forall.\ (B_{X_\alpha,t_\alpha,Z_\gamma})) \ \leq\ (E_{u_\theta,K_\tau} \times Y_\theta)$

(4) $\xrightarrow{(\times\leq)^r}_{alg\text{-}\alpha}$ $\ldots (-=(\bot \times \mu Z_\gamma.\forall.\mu.\forall.\ B_{X_\alpha,t_\alpha,Z_\gamma}),\ -=(E_{u_\theta,K_\tau} \times Y_\theta)) \rhd_\ell$
$\mu Z_\gamma.\forall.\mu.\forall.(B_{X_\alpha,t_\alpha,Z_\gamma}) \ \leq\ Y_\theta$

(5) $\xrightarrow{(\mathrm{RUnf}\leq)}_{alg\text{-}\alpha}$ $\ldots \rhd_\ell$
$\mu Z_\gamma.\forall.\mu.\forall.(B_{X_\alpha,t_\alpha,Z_\gamma}) \ \leq\ \mu Y_\beta.\forall u_\beta.\mu K_\vartheta.\forall.(E_{u_\beta,K_\vartheta} \times Y_\beta)$

(6) $\xrightarrow{(\forall\leq)}_{alg\text{-}\alpha}$ $\ldots (Z_\gamma = \mu Z_\gamma.\forall.\mu.\forall.(B_{X_\alpha,t_\alpha,Z_\gamma}),$
$Y_\beta = \mu Y_\beta.\forall u_\beta.\mu K_\vartheta.\forall.(D_{Y_\beta,u_\beta,K_\vartheta})),$
$(-, u_\beta) \leq (\top,\top) \rhd_\ell$
$\mu.\forall.(B_{X_\alpha,t_\alpha,Z_\gamma}) \leq\ \mu K_\vartheta.\forall.(E_{u_\beta,K_\vartheta} \times Y_\beta)$

(7) $\xrightarrow{(\forall\leq)}_{alg\text{-}\alpha}$ $\ldots (-=\mu.\forall.(B_{X_\alpha,t_\alpha,Z_\gamma}),\ K_\vartheta = \mu K_\vartheta.\forall.(E_{u_\beta,K_\vartheta} \times Y_\beta)),$
$(-,-) \leq (\top,\top) \rhd_\ell$
$(C_{X_\alpha,t_\alpha} \times Z_\gamma) \ \leq\ (E_{u_\beta,K_\vartheta} \times Y_\beta)$

(8) $\xrightarrow{(\times\leq)^l}_{alg\text{-}\alpha}$ $\ldots (-=C_{X_\alpha,t_\alpha} \times Z_\gamma,\ -=E_{u_\beta,K_\vartheta} \times Y_\beta) \rhd_\ell$
$(\bot \times\ t_\alpha) \times X_\alpha \ \leq\ (u_\beta \times \top) \times K_\vartheta$

(9) $\xrightarrow{(\times\leq)^r}_{alg\text{-}\alpha}$ $\ldots (-=(\bot \times\ t_\alpha) \times X_\alpha,\ -=(u_\beta \times \top) \times K_\vartheta) \rhd_\ell$
$X_\alpha \ \leq\ K_\vartheta$

(10) $\xrightarrow{(\mathrm{LUnf}\leq)}_{alg\text{-}\alpha}$ $\ldots \rhd_\ell$
$\mu X_\rho.\forall t_\rho.\ A_{X_\rho,t_\rho} \ \leq\ K_\vartheta$

(11) $\xrightarrow{(\mathrm{RUnf}\leq)}_{alg\text{-}\alpha}$ $\ldots \rhd_\ell$ (note that this is similar but not α-equivalent to the pre-judgement in 2, since the free variable u_β is different from u_θ)
$\mu X_\rho.\forall t_\rho.\ A_{X_\rho,t_\rho} \ \leq\ \mu K_\varphi.\forall.D_{Y_\beta,u_\beta,K_\varphi}$

(12) $\xrightarrow{(\forall\leq)}_{alg\text{-}\alpha}$ $\ldots (X_\rho = \mu X_\rho.\forall t_\rho.A_{X_\rho,t_\rho},\ \ K_\varphi = \mu K_\varphi.\forall.\ D_{Y_\beta,u_\beta,K_\varphi}),$
$(t_\rho, -) \leq (\top,\top) \rhd_\ell$
$\bot \times \mu Z_\varrho.\forall.\mu.\forall.\ B_{X_\rho,t_\rho,Z_\varrho} \ \leq\ E_{u_\beta,K_\varphi} \times Y_\beta$

(13) $\xrightarrow{(\times\leq)^r}_{alg\text{-}\alpha}$ $\ldots (-=(\bot \times \mu Z_\varrho.\forall.\mu.\forall.\ A_{X_\rho,t_\rho}),\ -=(E_{u_\beta,K_\varphi} \times Y_\beta)) \rhd_\ell$
$\mu Z_\varrho.\forall.\mu.\forall.B_{X_\rho,t_\rho,Z_\varrho} \ \leq\ Y_\beta$

(14) $\xrightarrow{(\mathrm{RUnf}\leq)}_{alg\text{-}\alpha}$ $\ldots \rhd_\ell$
$\mu Z_\varrho.\forall.\mu.\forall.B_{X_\rho,t_\rho,Z_\varrho} \ \leq\ \mu Y_\nu.\forall u_\nu.\mu K_\omega.\forall.(E_{u_\nu,K_\omega} \times Y_\nu)$

$$(15)\ \xrightarrow{(\forall\leq)}_{alg\text{-}\alpha}\ \ldots\ (Z_\varrho = \forall.\mu.\forall.(B_{X_\rho,t_\rho,Z_\varrho}),\ \ Y_\nu = \mu Y_\nu.\forall u_\nu.\mu K_\omega.\forall.(E_{u_\nu,K_\omega} \times Y_\nu),$$
$$(-, u_\nu) \leq (\top, \top) \rhd_\ell$$
$$\mu.\forall.B_{X_\rho,t_\rho,Z_\varrho} \leq \mu K_\omega.\forall.(E_{u_\nu,K_\omega} \times Y_\nu)$$
$$(16)\ \xrightarrow{(\forall\leq)}_{alg\text{-}\alpha}\ \ldots\ (- = \mu.\forall.(B_{X_\rho,t_\rho,Z_\varrho}),\ K_\omega = \mu K_\omega.\forall.(E_{u_\nu,K_\omega} \times Y_\nu)),$$
$$(-,-) \leq (\top, \top) \rhd_\ell$$
$$C_{X_\rho,t_\rho} \times Z_\varrho \leq E_{u_\nu,K_\omega} \times Y_\nu$$
$$(17)\ \xrightarrow{(\times\leq)^l}_{alg\text{-}\alpha}\ \ldots\ (- = C_{X_\rho,t_\rho} \times Z_\varrho,\ - = E_{u_\nu,K_\omega} \times Y_\nu) \rhd_\ell$$
$$(\bot \times t_\rho) \times X_\rho \leq (u_\nu \times \top) \times K_\omega$$
$$(18)\ \xrightarrow{(\times\leq)^r}_{alg\text{-}\alpha}\ \ldots\ (- = ((\bot \times t_\rho) \times X_\rho),\ - = ((u_\nu \times \top) \times K_\omega)) \rhd_\ell$$
$$X_\rho \leq K_\omega$$
$$(19)\ \xrightarrow{(\mathrm{LUnf}\leq)}_{alg\text{-}\alpha}\ \ldots \rhd_\ell$$
$$\mu X_\kappa.\forall t_\kappa.\ A_{X_\kappa,t_\kappa} \leq K_\omega$$
$$(20)\ \xrightarrow{(\mathrm{RUnf}\leq)}_{alg\text{-}\alpha}\ \ldots \rhd_\ell$$
$$\mu X_\kappa.\forall t_\kappa.\ A_{X_\kappa,t_\kappa} \leq \mu K_\iota.\forall.D_{Y_\nu,u_\nu,K_\iota}$$

As seen in the picture, starting from the eleventh judgement, every pair of compared types is similar to the one that has been met nine steps before. However, it always differs in some free variables. For example, if we consider the last nine steps, all the judgements from 12 to 17 contain a free variable t_ρ that is different from the variable t_α met nine steps before. Step 18 differs from 9 due to X_ρ and K_ω, which differ from X_α and K_ϑ . Step 19 differs from 10 due to K_ω/K_ϑ. Step 20 differs from 11 due to u_ν and Y_ν. No variable-to-variable comparison is performed; hence, once the first pair of similar comparisons is met, we know that the reduction will go on forever, with the same structure.

Remark 1 *Another kind of renaming could be used in the unfolding rules. When $X_{\alpha|\beta}$ is defined by $\mu X_\alpha.T$, our unfolding rule substitutes $X_{\alpha|\beta}$ with $(\mu X_\alpha.T) \uparrow \beta$, in accordance with the meaning we gave to recursive types. It would also be possible, however, to substitute $X_{\alpha|\beta}$ with just the body $T \uparrow \beta$. This alternative way of renaming corresponds to the following chain of equivalences:*

$$\mu X.T = \mu X.\,[T/X]\,T = \mu X.\,[T/X]\,([T/X]\,T)$$

rather than the following chain, which we exploit:

$$\mu X.T = [\mu X.T\,/\,X]\,T = [[\mu X.T\,/\,X]\,T\,/\,X]\,T \ldots$$

In this way, we would not create a new X variable. This variant has a greater chance of meeting an already met pair, since the outermost recursive variable

X is not renamed. We suspect that this variant is sound, but we did not prove this fact. Anyway, this variant is still not complete. Consider our judgement: every time we meet a pair of types that is similar to a previous one, it differs because of the free type variables, and this difference remains if we move to the variant algorithm. The only exceptions are the judgements in steps 18 and 19, which only contain free recursion variables. Indeed, with the variant algorithm, the X at step 9 would be the same as in step 18, while in the basic algorithm they are different variables. However, the K would still be different, since a new μK is generated every time Y is unfolded in step $9 * i + 5$. Hence, the variant algorithm is not complete either. It would be interesting to try and prove the soundness of this alternative algorithm, since it may be more convenient in some situations. We leave this as an open issue.

4.2 The Unsoundness of a Similarity Based Algorithm

The α-equivalence based algorithm diverges since, for some couples of pairs T, U and T', U', it fails to recognize that the second pair will behave like the first one; it exploits a notion of 'sameness' that is too strict.

Hence, it is natural to consider a weaker notion of sameness, and considering a pair to be the same as an already seen pair when the two are in fact only similar, according to the similarity relation $\simeq$. Thanks to the condition of face uniqueness (Definition 2.2), when we meet two similar types during subtype checking, this implies that they are "residuals" of the same subterm of the original judgement (or that they are both $\top$). This makes similarity quite a strong condition, and also makes it very efficient to check. If we are able to represent every type as a pointer to the subterm of the original judgement from where it comes, plus, possibly, some relabeling information, then similarity can be checked as pointer equality, which is the kind of efficiency we would need in actual implementations. Indeed, consider that before performing a reduction step in the subtype checking process, we have to compare the current pair with all the pairs in Π for sameness; hence the use of equality of pointer pairs instead of α-equivalence makes a dramatic difference.

Definition 4.1 *We say that $T \leq U \in_n^{\simeq} \Pi$ if the bi-environment Π contains at least n pairs $T'_1 \leq U'_1, \ldots T'_n \leq U'_n$ such that $T \simeq T'_i$ and $U \simeq U'_i$, for $i = 1 \ldots n$. Otherwise, we say that $T \leq U \notin_n^{\simeq} \Pi$.*

The next algorithm we consider is characterized by the following end rule.

$$\frac{T \leq U \in_1^{\simeq} \Pi \quad \Pi \vdash_\ell T, U \text{ Type}}{\Pi \vdash_\ell T \leq U} \quad (\text{End}^1_{\leq})$$

To preserve determinism, we define (without loss of generality) that this rule

takes priority over all the other rules except for the termination rules ($\mathrm{Id}_{\leq}$) and ($\top_{\leq}$).

In this section, for the sake of simplicity, we still consider an algorithm that records every met pair. It is easy to prove that our unsoundness proof still holds if the unfolding rules are the only ones that record the pairs met (actually, in this case the algorithm terminates only one step later, that is in step 10).

We call this new set of rules $\Re^{alg\text{-}1}$. They define a subtyping algorithm for recursive types that is complete, i.e. that always terminates.

The two relations $\xrightarrow{(R)}_{alg\text{-}1}$, $\twoheadrightarrow_{alg\text{-}1}$ are defined similarly to $\xrightarrow{(R)}_{\infty}$, $\twoheadrightarrow_{\infty}$.

In the following sections and in the proof of termination of $\Re^{alg\text{-}1}$ rules, we will make use of the following definition.

Definition 4.2 (TypesIn) $\mathrm{TypesIn}(\Gamma)$ *is the set of types contained in* Γ*; more precisely:*

$$\begin{aligned}
\mathrm{TypesIn}(()) &= \emptyset \\
\mathrm{TypesIn}(\Gamma, t_\alpha \leq T) &= \mathrm{TypesIn}(\Gamma) \cup \{T\} \\
\mathrm{TypesIn}(\Gamma, X_\alpha = T) &= \mathrm{TypesIn}(\Gamma) \cup \{T\} \\
\mathrm{TypesIn}(\Gamma, T) &= \mathrm{TypesIn}(\Gamma) \cup \{T\}
\end{aligned}$$

To prove termination of $\Re^{alg\text{-}1}$ rules, we also need the following lemma, where we prove that in any $\Re^{alg\text{-}1}$ reduction chain starting from $() \rhd_\ell T \leq U \in$ Start-J, all the types created are subexpressions, modulo $\simeq$ equivalence, of the initial types T and U. Hence, if we ignore labels, no new type is created during subtype comparison.

Lemma 4.3 *For each* $() \rhd_\ell T \leq U \in$ *Start-J, if*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg\text{-}1} \Pi' \rhd_\ell T' \leq U',$$

then one of the following two pairs of properties holds

$$(a) \begin{cases} \mathrm{TypesIn}(\mathrm{Left}(\Pi')) \cup \{T'\} \subseteq^{\simeq} \mathrm{SE}(T) \\ \mathrm{TypesIn}(\mathrm{Right}(\Pi')) \cup \{U'\} \subseteq^{\simeq} \mathrm{SE}(U) \end{cases}$$

or

$$(b) \begin{cases} \mathrm{TypesIn}(\mathrm{Left}(\Pi')) \cup \{T'\} \subseteq^{\simeq} \mathrm{SE}(U) \\ \mathrm{TypesIn}(\mathrm{Right}(\Pi')) \cup \{U'\} \subseteq^{\simeq} \mathrm{SE}(T) \end{cases}$$

Proof. By induction on the derivation length and by cases on the last applied

rule; we move between (a) and (b) when we consider $(\forall_{\leq})$ or $(\rightarrow_{\leq})$ as the last applied rule.□

Lemma 4.4 (Termination) *For each* $() \rhd_\ell T \leq U \in$ *Start-J, the* $\Re^{\text{alg-1}}$ *reduction process starting from* $() \rhd_\ell T \leq U$ *always terminates.*

Proof. The termination condition used in the end rule does not consider labels in the pairs that have already been met. Thus, using $\Re^{alg\text{-}1}$ rules, the reduction process must terminate because: a) modulo $\simeq$, only subexpressions of the initial types can be compared (Lemma 4.3); since initial types have a finite number of subexpressions, the unfolding rules can only be applied a finite number of times; b) termination of kernel Fun subtype checking implies that we cannot have an infinite reduction chain with a finite number of unfolding rule applications. If we had such an infinite reduction chain, then by starting from the last application of an unfolding rule we could obtain a judgement (with no recursive types) that makes the subtyping algorithm of standard kernel Fun diverge.□

It is not difficult to prove that the $\Re^{alg\text{-}1}$ algorithm has the same behavior as the one in [AC93] on first-order recursive types. But, if we consider second-order types, this algorithm is not sound with respect to the relation defined by $\Re^{\infty}$. We prove this fact by showing a subtyping pre-judgement that holds according to $\Re^{alg\text{-}1}$ but is not provable in $\Re^{\infty}$.

To simplify the presentation we still exploit the conventions used in the previous section, plus the following:

- A type $\mu X_\alpha.\forall t_\alpha.T$ where the variable X_α does not occur free in T, will be denoted by $\forall t_\alpha.T$.

The pre-judgement is $() \rhd_\ell \overline{T} \leq \overline{U}$, where

$$\overline{T} = \mu Z_\alpha.\forall t_\alpha.(\mu X_\beta.(X_\beta \times (t_\alpha \times Z_\alpha)))$$
$$\overline{U} = \mu.\forall u_\eta.\mu Y_\gamma.((\top \times (u_\eta \times (\mu.\forall v_\nu.Y_\gamma))) \times \top)$$

Note that these types have the same particular nested recursion μ-$\forall$-μ that characterized the pre-judgement studied in the previous section.

Using $\Re^{alg\text{-}1}$ rules, we have the following proof for our pre-judgement.

(1) $\quad () \rhd_\ell \overline{T} \leq \overline{U}$

(2) $\xrightarrow{(\forall_{\leq})}_{alg\text{-}1}$ $\quad (Z_\alpha = \overline{T}, - = \overline{U}),\ (t_\alpha, u_\eta) \leq (\top, \top) \rhd_\ell$
$\mu X_\beta.(X_\beta \times (t_\alpha \times Z_\alpha))$
$\leq \mu Y_\gamma.((\top \times (u_\eta \times (\mu.\forall v_\rho.Y_\gamma))) \times \top)$

(3) $\overset{(\times\leq)^l}{\longrightarrow}_{\textit{alg-1}}$ $\ldots (X_\beta = \mu X_\beta.(X_\beta \times (t_\alpha \times Z_\alpha)),$

$Y = \mu Y_\gamma.((\top \times (u_\eta \times (\mu.\forall v_\rho.Y_\gamma))) \times \top)) \rhd_\ell$

$X_\beta \leq (\top \times (u_\eta \times (\mu.\forall v_\rho.Y_\gamma)))$

(4) $\overset{(\mathrm{LUnf}_\leq)}{\longrightarrow}_{\textit{alg-1}}$ $\ldots X_\beta \leq (\top \times (u_\eta \times (\mu.\forall v_\rho.Y_\gamma))) \rhd_\ell$

$\mu X_\varphi.(X_\varphi \times (t_\alpha \times Z_\alpha)) \leq (\top \times (u_\eta \times (\mu.\forall v_\rho.Y_\gamma)))$

(5) $\overset{(\times\leq)^r}{\longrightarrow}_{\textit{alg-1}}$ $\ldots (X_\varphi = \mu X_\varphi.(X_\varphi \times (t_\alpha \times Z_\alpha)),$

$-\!= (\top \times (u_\eta \times (\mu.\forall v_\rho.Y_\gamma)))) \rhd_\ell$

$(t_\alpha \times Z_\alpha) \leq (u_\eta \times (\mu.\forall v_\rho.Y_\gamma))$

(6) $\overset{(\times\leq)^r}{\longrightarrow}_{\textit{alg-1}}$ [2] $\ldots (-\!= (t_\alpha \times Z_\alpha),\ -\!= (u_\eta \times (\mu.\forall v_\rho.Y_\gamma))\ ) \rhd_\ell$

$Z_\alpha \leq \mu.\forall v_\rho.Y_\gamma$

(7) $\overset{(\mathrm{LeftUnf}_\leq)}{\longrightarrow}_{\textit{alg-1}}$ $\ldots Z_\alpha \leq \mu.\forall v_\rho.Y_\gamma \rhd_\ell$

$\mu Z_\upsilon.\forall t_\upsilon.(\mu X_\tau.(X_\tau \times (t_\upsilon \times Z_\upsilon))) \leq \mu.\forall v_\rho.Y_\gamma$

(8) $\overset{(\forall\leq)}{\longrightarrow}_{\textit{alg-1}}$ $\ldots (Z_\upsilon = \mu Z_\upsilon.\forall t_\upsilon.\mu X_\tau.(X_\tau \times (t_\upsilon \times Z_\upsilon)),\ -\!= (\mu.\forall v_\rho.Y_\gamma)\ ),$

$(t_\upsilon, v_\rho) \leq (\top, \top) \rhd_\ell$

$\mu X_\tau.(X_\tau \times (t_\upsilon \times Z_\upsilon)) \leq Y_\gamma$

(9) $\overset{(\mathrm{RUnf}_\leq)}{\longrightarrow}_{\textit{alg-1}}$ $\ldots \mu X_\tau.(X_\tau \times (t_\upsilon \times Z_\upsilon)) \leq Y_\gamma \rhd_\ell$

$\mu X_\tau.(X_\tau \times (t_\upsilon \times Z_\upsilon)) \leq$

$\mu Y_\theta.((\top \times (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta))) \times \top)$

successful (since a similar pair is met in 2 and hence the rule $(\mathrm{End}^1_\leq)$ can be applied).

All the derivations not illustrated are trivially successful, so the pre-judgement is provable according to the $\Re^{\textit{alg-1}}$ rules. However, if we consider $\Re^\infty$, the reduction continues as follows.

(10). $\overset{(\times\leq)^l}{\longrightarrow}_\infty$ $\ldots (X_\tau = \mu X_\tau.(X_\tau \times (t_\upsilon \times Z_\upsilon)),$

$Y_\theta = \mu Y_\theta.((\top \times (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta))) \times \top)) \rhd_\ell$

$X_\tau \leq (\top \times (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta)))$

(11). $\overset{(\mathrm{LUnf}_\leq)}{\longrightarrow}_\infty$ $\ldots X_\tau \leq (\top \times (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta))) \rhd_\ell$

$\mu X_\kappa.(X_\kappa \times (t_\upsilon \times Z_\upsilon)) \leq (\top \times (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta)))$

[2] Note that the pre-judgement in 5 is also reduced to $\ldots \rhd t_\alpha \leq u_\eta$, which is true by $(\mathrm{Id}_\leq)$.

(12). $\overset{(\times\leq)^r}{\longrightarrow}_\infty$ $\ldots\ (X_\kappa{=}\mu X_\kappa.(X_\kappa \times (t_\upsilon \times Z_\upsilon)), -{=}(\top \times (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta)))) \rhd_\ell$
$(t_\upsilon \times Z_\upsilon) \leq (u_\eta \times (\mu.\forall v_\emptyset.Y_\theta))$

(13). $\overset{(\times\leq)^l}{\longrightarrow}_\infty$ $\ldots\ (-{=}(t_\upsilon \times Z_\upsilon), -{=}(u_\eta \times (\mu.\forall v_\emptyset.Y_\theta))) \rhd_\ell$
$t_\upsilon \leq u_\eta$

(14). $\overset{(\mathrm{VarTrans}_\leq)}{\longrightarrow}_\infty$ $\ldots \rhd_\ell \top \leq u_\eta$: fail!

Note that in 13, t_υ refers to the variable defined in step 8 where t_υ is unified with v_ρ. Then in 13 we cannot apply ($\mathrm{Id}_\leq$) as we did after step 5, so we have to apply ($\mathrm{VarTrans}_\leq$), thus obtaining the failure pre-judgement shown in 14.

The above derivation shows us that, in order to define a sound stop criterion, correct unification of type variables is the key issue. In Section 5 we will prove that the stop conditions we propose in the next section guarantee some properties of variable unification that imply the correctness of the resulting algorithm.

4.3 A Sound and Complete Algorithm

We may say that the $\Re^{alg\text{-}1}$ algorithm was too eager to terminate; had it waited a bit longer, it would have discovered the pending failure. Indeed, we stated that stopping when a pre-judgement is met (modulo similarity) the third time is a sound ending criterion that defines a sound algorithm.

The rules of our sound and complete algorithm are obtained by adding two new termination rules to the $\Re^\infty$ system and by modifying the unfolding rules as follows.

$$\frac{X_{\alpha|\beta} \leq U \ \in_2^\simeq \Pi \quad \Pi \vdash_\ell X_{\alpha|\beta}, U \ \mathrm{Type}}{\Pi \vdash_\ell X_{\alpha|\beta} \ \leq \ U} \quad (\mathrm{LEnd}_\leq)$$

$$\frac{T \ \leq \ Y_{\nu|\eta} \ \in_2^\simeq \Pi \quad \Pi \vdash_\ell T, Y_{\nu|\eta} \ \mathrm{Type}}{\Pi \vdash_\ell T \ \leq \ Y_{\nu|\eta}} \quad (\mathrm{REnd}_\leq)$$

$$\frac{\begin{array}{l} X_{\alpha|\beta} \leq U \notin_2^\simeq \Pi \\ X_\alpha{=}T \in \mathrm{Left}(\Pi) \quad \Pi, X_{\alpha|\beta} \leq U \vdash_\ell T \uparrow \beta \ \leq \ U \end{array}}{\Pi \vdash_\ell X_{\alpha|\beta} \ \leq \ U} \quad (\mathrm{LUnf}^2_\leq)$$

$$\frac{\begin{array}{ll} T \leq Y_{\nu|\eta} \not\in_2^{\simeq} \Pi & \textit{for all } X_{\theta|\delta}.\ (T \neq X_{\theta|\delta}) \\ Y_\nu{=}U \in \mathrm{Right}(\Pi) & \Pi, T \leq Y_{\nu|\eta} \vdash_\ell T \ \leq\ U \uparrow \eta \end{array}}{\Pi \vdash_\ell T \ \leq\ Y_{\nu|\eta}} \quad (\mathrm{RUnf}^2_{\leq})$$

To ensure termination, the ($\_\mathrm{End}_{\leq}$) rules take priority over rules ($\_\mathrm{Unf}^2_{\leq}$) and ($\mathrm{VarTrans}_{\leq}$) — we make this explicit in the first premises of the ($\_\mathrm{Unf}^2_{\leq}$) rules above. To preserve determinism, we also define that ($\top_{\leq}$) has priority over ($\_\mathrm{End}_{\leq}$).

With this extension we have a new set of rules that we call $\Re^{alg\text{-}2}$ and whose backward application defines a subtyping algorithm for recursive types. We outline the termination proof in the next section. Hereafter we will indicate with $\Pi \vdash_\ell^{alg\text{-}2} T \leq U$ the fact that $\Pi \vdash_\ell T \leq U$ holds with respect to $\Re^{alg\text{-}2}$ rules.

4.4 Termination, Invariance of Good Formation and Non Necessity of Renaming

In this section we prove some basic properties of systems $\Re^{\infty}$ and $\Re^{alg\text{-}2}$, namely:

- the algorithm defined by $\Re^{alg\text{-}2}$ always terminates;
- the backward application of both $\Re^{alg\text{-}2}$ and $\Re^{\infty}$ rules reduces a well-formed subtyping pre-judgement to a collection of pre-judgements that are still well-formed;
- variable renaming is not necessary in the $\Re^{alg\text{-}2}$ and in the $\Re^{\infty}$ system.

We will show later how to transform any $\Re^{alg\text{-}2}$ into an $\Re^{\infty}$ proof, using a stepwise process that produces intermediate proofs where some unfoldings are performed according to the $\Re^{\infty}$ rule, and some are stopped according to the $\Re^{alg\text{-}2}$ rule. For this reason we define a system $\Re^{\mathrm{alg}|\infty}$ where both the ($\_\mathrm{Unf}^2_{\leq}$) and the ($\_\mathrm{Unf}_{\leq}$) rules can be used, and where proofs can be finite or infinite; hence, $\Re^{\mathrm{alg}|\infty}$ proofs are a superset of both $\Re^{alg\text{-}2}$ and $\Re^{\infty}$ proofs. $\Re^{\mathrm{alg}|\infty}$ induces two relations $\xrightarrow{(R)}_{\mathrm{alg}|\infty}$ and $\twoheadrightarrow_{\mathrm{alg}|\infty}$, that relate a pre-judgement to those that can be obtained by the backward application of a rule R ($\xrightarrow{(R)}_{\mathrm{alg}|\infty}$) or by a sequence of zero or more rule applications ($\twoheadrightarrow_{\mathrm{alg}|\infty}$).

Definition 4.5 *We define Start-J$^{alg|\infty}$ as the set of all pre-judgements that can be reached starting from a Start-J pre-judgement by using $\twoheadrightarrow_{alg|\infty}$.*

Of course, Start-J$^{\mathrm{alg}|\infty}$ is a superset of both Start-J$^{alg\text{-}2}$ and Start-J$^{\infty}$.

4.4.1 Termination

To prove termination of $\Re^{alg\text{-}2}$ rules, we restate Lemma 4.3 for $\Re^{alg\text{-}2}$ and $\Re^{\infty}$ reduction chains starting from pre-judgements $() \rhd_\ell T \leq U \in$ Start-J.

Lemma 4.6 *For each* $() \rhd_\ell T \leq U \in$ *Start-J, if*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi' \rhd_\ell T' \leq U',$$

then one of the following two pairs of properties holds

$$(a) \begin{cases} \text{TypesIn}(\text{Left}(\Pi')) \cup \{T'\} \subseteq^{\simeq} \text{SE}(T) \\ \text{TypesIn}(\text{Right}(\Pi')) \cup \{U'\} \subseteq^{\simeq} \text{SE}(U) \end{cases}$$

or

$$(b) \begin{cases} \text{TypesIn}(\text{Left}(\Pi')) \cup \{T'\} \subseteq^{\simeq} \text{SE}(U) \\ \text{TypesIn}(\text{Right}(\Pi')) \cup \{U'\} \subseteq^{\simeq} \text{SE}(T) \end{cases}$$

Proof. By induction on the derivation length and by cases on the last applied rule; we move between (a) and (b) when we consider $(\forall_{\leq})$ or $(\rightarrow_{\leq})$ as the last applied rule.□

Property 4.7 (Termination) *For each* $() \rhd_\ell T \leq U \in$ *Start-J, the* $\Re^{\text{alg-2}}$ *reduction process starting from* $() \rhd_\ell T \leq U$ *always terminates.*

Proof. Use Lemma 4.6 and proceed as in Lemma 4.4.□

4.4.2 Invariance of good formation

In this section we prove that our renaming technique guarantees that the backward application of $\Re^{\text{alg}|\infty}$ rules preserves good formation; we also prove a couple of quite technical lemmas, like the first one below.

Lemma 4.8 *For each* $() \rhd_\ell T \leq U \in$ *Start-J, if*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi' \rhd_\ell T' \leq U',$$

then

1. $T', U' \in$*LR-Types*
2. $\text{FV}(T') \subseteq \text{Def}(\text{Left}(\Pi'))$, $\text{FV}(U') \subseteq \text{Def}(\text{Right}(\Pi'))$
3. $\forall\, (t_\alpha, u_\beta) \leq (T'', U'') \in \Pi'$.

3.0 $\mathrm{FV}(T'') \subseteq \mathrm{Def}(\mathrm{Left}(\Pi'))$, $\mathrm{FV}(U'') \subseteq \mathrm{Def}(\mathrm{Right}(\Pi')$

3.1 $T'', U'' \in$ *LR-Types*

3.2 $\mathrm{Root}(T'') = \alpha.0$, $\mathrm{Root}(U'') = \beta.0$

3.3 $\alpha \prec \mathrm{Root}(T')$, $\beta \prec \mathrm{Root}(U')$

4. $\forall\, (X_\alpha = T'',\ Y_\beta = U'') \in \Pi'$.

4.0 $\mathrm{FV}(T'') \subseteq \mathrm{Def}(\mathrm{Left}(\Pi'))$, $\mathrm{FV}(U'') \subseteq \mathrm{Def}(\mathrm{Right}(\Pi')$

4.1 $T'', U'' \in$ *LR-Types*

4.2 $\mathrm{Root}(T'') = \alpha$, $\mathrm{Root}(U'') = \beta$

4.3 $\alpha \prec \mathrm{Root}(T')$, $\beta \prec \mathrm{Root}(U')$

Proof. Points 1, 2, 3 and 4 are simultaneously proved by induction on the derivation length and by cases on the last applied rule.
The only interesting cases are (VarTrans$_\leq$) and (_Unf$_\leq$). We consider only the proof of properties 3.3 and 4.3; the other ones are obvious. We have to prove that if the lemma holds for $\Pi' \rhd_\ell t_{\vartheta|\nu} \leq U'$ and $t_\vartheta \leq T' \in \mathrm{Left}(\Pi')$, then the lemma also holds for $\Pi' \rhd_\ell T' \leq U'$. To do this, we observe that, by hypothesis, we have $t_{\vartheta|\nu} \in$LR-Types and then $\nu = \vartheta.1.\nu'$; moreover, we have $\mathrm{Root}(T') = \vartheta.0$ (by the induction hypothesis 3.2) and $\mathrm{Root}(t_{\vartheta|\nu}) = \nu = \vartheta.1.\nu'$. Therefore, $\mathrm{Root}(t_{\vartheta|\nu}) \prec_{rl} \mathrm{Root}(T')$, hence $\mathrm{Root}(t_{\vartheta|\nu}) \prec \mathrm{Root}(T')$. Now the proof follows by induction hypothesis and by transitivity of $\prec$. Case (_Unf$_\leq$) is similar, but $\prec_p$ is used instead of $\prec_{rl}$.□

This lemma has some interesting corollaries.

Corollary 4.9 *For each* $() \rhd_\ell T \leq U \in$ *Start-J, if*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi' \rhd_\ell T' \leq U',$$

and

$$\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n$$

is the sequences of the labels of the type variables defined in $\mathrm{Left}(\Pi')$*, while*

$$\beta_1, \beta_2, \ldots, \beta_{m-1}, \beta_m$$

is the sequence of recursion variables defined in $\mathrm{Left}(\Pi')$*, then*

$$\alpha_1 \prec \alpha_2 \prec \ldots \prec \alpha_{n-1} \prec \alpha_n \qquad \beta_1 \prec \beta_2 \prec \ldots \prec \beta_{m-1} \prec \beta_m$$

The same holds for $\mathrm{Right}(\Pi')$.

Proof. By induction on the derivation length and by cases on the last applied rule. If we suppose that the corollary holds for $\Pi'' \rhd_\ell T'' \leq U''$ such that

$$() \rhd_\ell T \leq U \twoheadrightarrow_{\mathrm{alg}|\infty} \Pi'' \rhd_\ell T'' \leq U'' \overset{(R)}{\rightarrow} \Pi' \rhd_\ell T' \leq U'$$

then the conclusion follows by 4.8 and by observing that Π' may differ from Π'' only in terms of the last element inserted by the application of rule (R).□

This corollary implies that no variable is defined twice in Π'; indeed, each label α_i is different from the others.

Corollary 4.10 *For each* $() \rhd_\ell T \leq U \in$ *Start-J, if*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi' \rhd_\ell T' \leq U',$$

then

$$\mathrm{Def}(\mathrm{Left}(\Pi')) \cap \mathrm{DV}(T') = \emptyset$$
$$\mathrm{Def}(\mathrm{Right}(\Pi')) \cap \mathrm{DV}(U') = \emptyset$$

Proof. By contradiction and using Lemma 4.8 and Lemma 3.19.□

Corollary 4.11 *For each* $\Pi' \rhd_\ell T' \leq U' \in$ *Start-*$J^{alg|\infty}$:

1. $\Pi' = \Pi'', (X_\alpha = T'', Y_\beta = U''), \Pi''' \Rightarrow$
   $X_\alpha \notin (\mathrm{Def}(\mathrm{Left}(\Pi'')) \cup \mathrm{Def}(\mathrm{Left}(\Pi''')))$
   $T'' = \mu X_\alpha.A \to B$ *or* $T'' = \mu X_\alpha.\forall t_\alpha \leq A.B$
   $\mathrm{DV}(T'') \cap \mathrm{Def}(\mathrm{Left}(\Pi'')) = \emptyset,\ \mathrm{FV}(T'') \subseteq \mathrm{Def}(\mathrm{Left}(\Pi''))$
   $Y_\beta \notin (\mathrm{Def}(\mathrm{Right}(\Pi'')) \cup \mathrm{Def}(\mathrm{Right}(\Pi''')))$
   $U'' = \mu Y_\beta.A' \to B'$ *or* $U'' = \mu Y_\beta.\forall u_\beta \leq A'.B'$
   $\mathrm{DV}(U'') \cap \mathrm{Def}(\mathrm{Right}(\Pi'')) = \emptyset,\ \mathrm{FV}(U'') \subseteq \mathrm{Def}(\mathrm{Right}(\Pi''))$

2. $\Pi' = \Pi'', (t_\alpha, u_\beta) \leq (T'', U''), \Pi''' \Rightarrow$
   $t_\alpha \notin (\mathrm{Def}(\mathrm{Left}(\Pi'')) \cup \mathrm{Def}(\mathrm{Left}(\Pi''')))$
   $\mathrm{DV}(T'') \cap \mathrm{Def}(\mathrm{Left}(\Pi'')) = \emptyset,\ \mathrm{FV}(T'') \subseteq \mathrm{Def}(\mathrm{Left}(\Pi''))$
   $u_\beta \notin (\mathrm{Def}(\mathrm{Right}(\Pi'')) \cup \mathrm{Def}(\mathrm{Right}(\Pi''')))$
   $\mathrm{DV}(U'') \cap \mathrm{Def}(\mathrm{Right}(\Pi'')) = \emptyset,\ \mathrm{FV}(U'') \subseteq \mathrm{Def}(\mathrm{Right}(\Pi''))$

3. $\Pi' = \Pi'', T'' \diamond U'', \Pi''' \Rightarrow$
$\mathrm{DV}(T'') \cap \mathrm{Def}(\mathrm{Left}(\Pi'')) = \emptyset,\ \mathrm{FV}(T'') \subseteq \mathrm{Def}(\mathrm{Left}(\Pi''))$
$\mathrm{DV}(U'') \cap \mathrm{Def}(\mathrm{Right}(\Pi'')) = \emptyset,\ \mathrm{FV}(U'') \subseteq \mathrm{Def}(\mathrm{Right}(\Pi''))$

Proof. By induction on the derivation length and by applying Lemma 4.8 and Corollary 4.9.□

Lemma 4.12 *Let* $\Pi' \rhd_\ell T' \leq U' \in$ *Start-*$J^{alg|\infty}$. *If* $\overline{\Gamma} = \mathrm{Left}(\Pi')$ *and* $\overline{T} = T'$ *(or* $\overline{\Gamma} = \mathrm{Right}(\Pi')$ *and* $\overline{T} = U'$*), then the pre-judgements* $\overline{\Gamma} \rhd_\ell \overline{T}$ *Type and* $\overline{\Gamma} \rhd_\ell$ *Env are provable by the backward application of the good formation rules if they enjoy the following properties:*

WF1: $\mathrm{DV}(\overline{T}) \cap \mathrm{Def}(\overline{\Gamma}) = \emptyset,\ \ \mathrm{FV}(\overline{T}) \subseteq \mathrm{Def}(\overline{\Gamma})$;

WF2: $\forall\ U \in\ \mathrm{TypesIn}(\overline{\Gamma})$

$$\overline{\Gamma} = \overline{\Gamma}', X_\alpha = U, \overline{\Gamma}'' \Rightarrow \begin{cases} X_\alpha \notin \mathrm{Def}(\overline{\Gamma}') \\ U = \mu X_\alpha.A \to B \ \textit{or}\ U = \mu X_\alpha.\forall t_\alpha \leq A.B \ \textit{or} \\ \textit{or}\ U = \top_\alpha \\ \mathrm{DV}(U) \cap \mathrm{Def}(\overline{\Gamma}') = \emptyset \\ \mathrm{FV}(U) \subseteq \mathrm{Def}(\overline{\Gamma}') \end{cases}$$

$$\overline{\Gamma} = \overline{\Gamma}', u_\alpha \leq U, \overline{\Gamma}'' \ \Rightarrow \begin{cases} u_\alpha \notin (\mathrm{Def}(\overline{\Gamma}') \cup \mathrm{Def}(\overline{\Gamma}'')) \\ \mathrm{DV}(U) \cap \mathrm{Def}(\overline{\Gamma}') = \emptyset \\ \mathrm{FV}(U) \subseteq \mathrm{Def}(\overline{\Gamma}') \end{cases}$$

$$\overline{\Gamma} = \overline{\Gamma}', U, \overline{\Gamma}'' \qquad \Rightarrow \mathrm{DV}(U) \cap \mathrm{Def}(\overline{\Gamma}') = \emptyset,\ \mathrm{FV}(U) \subseteq \mathrm{Def}(\overline{\Gamma}')$$

Proof. We first observe that backward application of good formation rules always terminates, and preserves properties WF1 and WF2.

Then, we observe that if WF1 and WF2 hold for a good formation pre-judgement, then rules (Γ BoundForm), (T-Var Form), (Γ EqForm) and (R-VarForm) are always applicable. This proves the claim, since these rules are the only ones that can fail. □

It is not difficult to prove that the two corollaries 4.10 and 4.11, derived from 4.8, imply properties WF1 and WF2 for each Start-$\mathrm{J}^{\mathrm{alg}|\infty}$ judgement. In other words, backward application of $\Re^{\mathrm{alg}|\infty}$ rules preserves good formation.

Hence, in the premises of rules ($\mathrm{Id}_{\leq}$), ($\top_{\leq}$), ($\mathrm{LEnd}_{\leq}$) and ($\mathrm{REnd}_{\leq}$) good formation judgements are superfluous and thus can be eliminated.

Notation 4.13 *Hereafter,* $\Pi \vdash_\ell T \leq U$ *and* $\Pi \vdash_{\ell-NWF} T \leq U$ *will respectively indicate the fact that the pre-judgement* $\Pi \rhd_\ell T \leq U$ *is proved by* $\Re^{alg|\infty}$ *rules by considering and by not considering good formation checking.*

Property 4.14 *For each* $\Pi \rhd_\ell T \leq U \in$ *Start-*$J^{alg|\infty}$*:*

$$\Pi \vdash_\ell T, U \text{ Type} \Rightarrow (\Pi \vdash_\ell T \leq U \Leftrightarrow \Pi \vdash_{\ell-NWF} T \leq U)$$

For this reason, hereafter we will ignore good formation rules when we analyze judgements belonging to Start-$\mathrm{J}^{\mathrm{alg}|\infty}$.

4.4.3 Non necessity of renaming

In this section we give an important property of our system, the 'non necessity' of renaming in both $\Re^{\infty}$ and $\Re^{alg\text{-}2}$ systems. Some of the properties we prove in this section will be needed to prove the correctness and transitivity of $\Re^{alg\text{-}2}$ system.

Lemma 4.15 (Similarity of bounds) *For each* $() \rhd_\ell T \leq U \in$ *Start-J,* $\Pi' \rhd_\ell T' \leq U'$ *and* $\Pi'' \rhd_\ell T'' \leq U''$ *such that*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi' \rhd_\ell T' \leq U' \text{ and}$$
$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi'' \rhd_\ell T'' \leq U''$$

we have:

$$X_{\alpha'} = V' \in \mathrm{Left}(\Pi') \text{ and } X_{\alpha''} = V'' \in \mathrm{Left}(\Pi'') \Rightarrow V' \simeq V''$$

and

$$t_{\alpha'} \leq V' \in \mathrm{Left}(\Pi') \text{ and } t_{\alpha''} \leq V'' \in \mathrm{Left}(\Pi'') \Rightarrow V' \simeq V''$$

and the same holds when Left is substituted by Right.

Proof. By induction on the derivation length and by applying Lemma 4.6 and the uniqueness of variable faces in T and U.□

The following lemma is not immediate, but is crucial in order to prove both the non necessity of labels (Corollary 4.17) and the correctness theorem.

Lemma 4.16 *For each* $\Pi' \rhd_\ell T' \leq U' \in$ *Start-*$J^{alg|\infty}$*, if* $\chi \in \mathrm{FV}(T')$ *and*

$\Gamma' = \mathrm{Left}(\Pi')$ *(*$\chi \in \mathrm{FV}(U')$ *and* $\Gamma' = \mathrm{Right}(\Pi')$*), then:*

1. $\Gamma' = \Gamma_1,\, X_\alpha = (\mu X_\alpha.\forall \chi \leq A'.A),\, \chi \leq A', \Gamma_2$ *or*
   $\Gamma' = \Gamma_1,\, \chi = A, \Gamma_2$
2. $\mathrm{TypesIn}(\Gamma_2) \cup \{T'\} \subseteq^{\simeq} \mathrm{SE}(A)$ $\quad (\mathrm{TypesIn}(\Gamma_2) \cup \{U'\} \subseteq^{\simeq} \mathrm{SE}(A))$

Proof. We consider only the case $\chi \in \mathrm{FV}(T')$ and $\Gamma' = \mathrm{Left}(\Pi')$, the second one has the same proof. Both points 1 and 2 follow by induction on the number of the reduction steps needed to obtain $\Pi' \rhd_\ell T' \leq U'$, and by cases on the last applied rule. Point 1 does not pose any particular problem, so we omit the proof. We prove point 2 by considering only the most interesting cases, i.e. cases ($\mathrm{VarTrans}_{\leq}$) and ($\mathrm{LUnf}_{\leq}$).

- Case ($\mathrm{VarTrans}_{\leq}$). We have to prove that if the lemma holds for $\Pi' \rhd_\ell t_{\alpha|\beta} \leq U'$ and for each $\Pi'' \rhd_\ell T'' \leq U''$ such that

  $$() \rhd_\ell T \leq U \;\twoheadrightarrow_{\mathrm{alg}|\infty}\; \Pi'' \rhd_\ell T'' \leq U'' \;\twoheadrightarrow_{\mathrm{alg}|\infty}\; \Pi' \rhd_\ell t_{\alpha|\beta} \leq U',$$

  then it also holds for $\Pi' \rhd_\ell T' \leq U'$, where $t_\alpha \leq T' \in \mathrm{Left}(\Pi')$. For this purpose we observe the following facts.
  By the induction hypothesis, we have

  $$\mathrm{Left}(\Pi') = \Gamma' = (\Gamma_1,\, X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T', \Gamma_2) \quad \text{(a1)}$$
  $$\mathrm{TypesIn}(\Gamma_2) \cup \{t_\alpha\} \subseteq^{\simeq} \mathrm{SE}(A). \quad \text{(a2)}$$

  By the induction hypothesis, the lemma holds for the pre-judgement

  $$\Pi_1 \rhd_\ell (\mu X_\alpha.\forall t_\alpha \leq T'.A) \leq (\mu X'_{\alpha'}.\forall t'_{\alpha'} \leq \overline{T}'.\overline{T}'')$$

  that inserts t_α and X_α (the recursion variable that is defined together with t_α) in Γ'. This means that Π_1 is the part of Π' corresponding to Γ_1 ($\Gamma_1 = \mathrm{Left}(\Pi_1)$) and that, by induction, for each $\chi' \in \mathrm{FV}(\mu X_\alpha.\forall t_\alpha \leq T'.A)$:

  $$\Gamma_1 \;=\; \Gamma_{1.1},\, Y_\beta = (\mu Y_\beta.\forall \chi' \leq B'.B),\, \chi' \leq B', \Gamma_{1.2} \;\text{ or}$$
  $$\Gamma_1 \;=\; \Gamma_{1.1},\, \chi' = B,\, \Gamma_{1.2}$$
  $$\mathrm{TypesIn}(\Gamma_{1.2}) \cup \{\mu X_\alpha.\forall t_\alpha \leq T'.A\} \subseteq^{\simeq} \mathrm{SE}(B). \quad (b)$$

  Moreover: $(\mathrm{FV}(T') - \{X_\alpha\}) \subseteq \mathrm{FV}(\mu X_\alpha.\forall t_\alpha \leq T'.A)$ $\quad (c)$
  At this stage of the proof we have the following possible decompositions

for $\Gamma' = \mathrm{Left}(\Pi')$:

$$\begin{aligned}\Gamma' \;&= \Gamma_{1.1},\, Y_\beta = (\mu Y_\beta.\forall\chi' \leq B'.B),\, \chi' \leq B',\\ &\qquad \Gamma_{1.2},\; X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T', \Gamma_2 \;\text{ or}\\ \Gamma' \;&= \;\Gamma_{1.1},\, \chi' = B,\, \Gamma_{1.2},\, X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T', \Gamma_2\end{aligned}$$

Now, let $\chi \in \mathrm{FV}(T')$. If $\chi \neq X_\alpha$ then by (c) we have $\chi \in \mathrm{FV}(\mu X_\alpha.\forall t_\alpha \leq T'.A)$ and we can consider the two possible decompositions of Γ_1 given above where $\chi = \chi'$. Hence, for both decompositions, we have to prove

$$\mathrm{TypesIn}(\Gamma_{1.2},\; X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T', \Gamma_2) \cup \{T'\} \subseteq^{\simeq} \mathrm{SE}(B)$$

To this end we first observe that

$$\begin{aligned}&\mathrm{TypesIn}(\Gamma_{1.2},\; X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T', \Gamma_2) \;=\\ &\quad \mathrm{TypesIn}(\Gamma_{1.2},\; X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T') \cup\; \mathrm{TypesIn}(\Gamma_2)\end{aligned}$$

Then we observe that from (b) and $T' \in \mathrm{SE}(\mu X_\alpha.\forall t_\alpha \leq T'.A)$ we have

$$\mathrm{TypesIn}(\Gamma_{1.2},\; X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\, t_\alpha \leq T') \cup \{T'\} \subseteq^{\simeq} \mathrm{SE}(B)$$

while $\mathrm{TypesIn}(\Gamma_2) \subseteq^{\simeq} \mathrm{SE}(B)$ follows by

$$\begin{aligned}&\mathrm{TypesIn}(\Gamma_2) \subseteq^{\simeq} \mathrm{SE}(A) && \text{by (a2)}\\ &\mathrm{SE}(A) \subseteq^{\simeq} \mathrm{SE}(\mu X_\alpha.\forall t_\alpha \leq T'.A) && \text{by def.}\\ &\mathrm{SE}(\mu X_\alpha.\forall t_\alpha \leq T'.A) \subseteq^{\simeq} \mathrm{SE}(B) && \text{by (b)}\end{aligned}$$

If $\chi = X_\alpha$, since we can have only one definition of X_α in Γ' (Corollary 4.9), we have the following decomposition for Γ':

$$\Gamma' \;=\; \Gamma_1, X_\alpha = (\mu X_\alpha.\forall t_\alpha \leq T'.A),\; t_\alpha \leq T', \Gamma_2$$

In this case the thesis to prove is

$$\mathrm{TypesIn}(\, t_\alpha \leq T', \Gamma_2) \cup \{T'\} \subseteq^{\simeq} \mathrm{SE}(\mu X_\alpha.\forall t_\alpha \leq T'.A)$$

which directly follows by the induction hypothesis (a2) and by observing that $T' \in \mathrm{SE}(\mu X_\alpha.\forall t_\alpha \leq T'.A)$.

- Case ($\mathrm{LUnf}_{\leq}$). The proof is similar to the previous case. We have to prove that if the lemma holds for $\Pi' \rhd_\ell X_{\alpha|\beta} \leq U'$ and for each $\Pi'' \rhd_\ell T'' \leq U''$ such that

$$() \rhd_\ell T \leq U \;\; \twoheadrightarrow_{\mathrm{alg}|\infty} \;\; \Pi'' \rhd_\ell T'' \leq U'' \;\; \twoheadrightarrow_{\mathrm{alg}|\infty} \;\; \Pi' \rhd_\ell X_{\alpha|\beta} \leq U'$$

then it also holds for $\Pi', X_{\alpha|\beta} \leq U' \rhd_\ell T' \leq U'$ where $T' = A \uparrow \beta$ and $X_\alpha = A \in \mathrm{Left}(\Pi')$. By Lemma 4.11 is $A = \mu X_\alpha.A' \to A''$ or $A = \mu X_\alpha.\forall t_\alpha \leq A'.A''$. Suppose we are in the first case (for the second one the proof is the same). We observe that, by the induction hypothesis, we have

$$\mathrm{Left}(\Pi') \;=\; \Gamma' \;=\; (\Gamma_1,\, X_\alpha = A, \Gamma_2)$$
$$\mathrm{TypesIn}(\Gamma_2) \cup \{X_{\alpha|\beta}\} \subseteq^\simeq \mathrm{SE}(A) \qquad (a)$$

By the induction hypothesis, we also have that the lemma holds for the pre-judgement

$$\Pi_1 \rhd_\ell A \leq\; \mu X'_{\alpha'}.\overline{T}. \to \overline{\overline{T}}$$

that inserts X_α in Γ' (recall that $A = \mu X_\alpha.A' \to A''$). As in the previous case, this means that $\Gamma_1 = \mathrm{Left}(\Pi_1)$ and that for each $\chi' \in \mathrm{FV}(A)$:

$$\Gamma_1 \;=\; \Gamma_{1.1},\, Y_\beta = (\mu Y_\beta.\forall \chi' \leq B'.B),\, \chi' \leq B', \Gamma_{1.2} \;\text{ or}$$
$$\Gamma_1 \;=\; \Gamma_{1.1},\, \chi' = B, \Gamma_{1.2}$$
$$\mathrm{TypesIn}(\Gamma_{1.2}) \cup \{A\} \subseteq^\simeq \mathrm{SE}(B). \qquad (b)$$

Moreover,

$$\mathrm{FV}(A) = \mathrm{FV}(T') \;(c)$$
$$T' \simeq A. \qquad (d)$$

Now we have to prove that the lemma holds for $\Pi', X_{\alpha|\beta} \leq U' \rhd_\ell T' \leq U'$. Let $\chi \in \mathrm{FV}(T')$. By (c) we have $\chi \in \mathrm{FV}(A)$ and, as in the previous case, we can consider the two possible decompositions of Γ_1 given above where $\chi = \chi'$. Hence, for both decompositions, we have to prove

$$\mathrm{TypesIn}(\Gamma_{1.2},\; X_\alpha = A, \Gamma_2,\; X_{\alpha|\beta}) \cup \{T'\} \subseteq^\simeq \mathrm{SE}(B)$$

By (b) and (d) we already have

$$\mathrm{TypesIn}(\Gamma_{1.2},\; X_\alpha = A) \cup \{T'\} \subseteq^\simeq \mathrm{SE}(B)$$

and in particular

$$A \in^\simeq \mathrm{SE}(B)$$

which implies $\mathrm{SE}(A) \subseteq^\simeq \mathrm{SE}(B)$ (e).
Hence, to conclude the proof, it remains to prove that

$$\mathrm{TypesIn}(\Gamma_2,\; X_{\alpha|\beta}) \subseteq^\simeq \mathrm{SE}(B)$$

This directly follows by (a) and (e).□

By using Lemma 4.16, we prove below that for each pre-judgement $\Pi' \rhd_\ell T' \leq U'$ in Start-$J^{alg|\infty}$, and for any χ free in the compared types, no other variable with the same face (even if with a different label) is defined after the definition of χ in the bi-environment. This property implies that labels can be ignored during subtype checking, since the correct definition of a defined variable can be identified simply by looking for the most recent one with the same face.

This is, in practice, an extremely useful result. During subtype checking, renaming is needed, in principle, to avoid variable capture. In a typical implementation, when a type is renamed, memory has to be allocated to store the newly created type . From the experience gained in the study and realization of languages such as Galileo [ACO85] and Fibonacci [AGO95] we learned that memory allocation has to be reduced as far as possible to improve the efficiency of type checking. So, avoiding renaming is already an effective way to improve the efficiency of our algorithm. Moreover, as already stated, if variables are never renamed, then type similarity can be checked by pointer equality, hence we have an efficient way of establishing when the same pair of types is compared twice during subtyping checking; as we have seen, this is the crucial operation to optimize when recursive types are compared.

Corollary 4.17 (Irrelevance of labels) *For each* $() \rhd_\ell T \leq U \in$ *Start-J and* $\Pi' \rhd_\ell T' \leq U'$ *such that*

$$() \rhd_\ell T \leq U \twoheadrightarrow_{alg|\infty} \Pi' \rhd_\ell T' \leq U',$$

if $\chi \in \mathrm{FV}(T')$ *and* $\Gamma = \mathrm{Left}(\Pi')$ *(if* $\chi \in \mathrm{FV}(U')$ *and* $\Gamma = \mathrm{Right}(\Pi')$*) then:*

1. $\Gamma = \Gamma',\ X_\alpha = (\mu X_\alpha.\forall \chi \leq A.A'),\ \chi \leq A,\ \Gamma''$ *or* $\Gamma = \Gamma',\ \chi = A,\ \Gamma''$
2. $\chi \not\in^{\simeq} \mathrm{Def}(\Gamma'')$

Proof. Both points 1 and 2 follow by induction on the number of the reduction steps needed to obtain $\Pi' \rhd_\ell T' \leq U'$, and by cases on the last applied rule. Point 1 does not pose any particular problem, so we omit the proof. To prove (2), we first observe that

$$s_\nu \in \mathrm{Def}(\Gamma'') \Rightarrow \Gamma'' = \Gamma_1,\ (Z_\nu = \mu Z_\nu.\forall s_\nu \leq B.B'),\ s_\nu \leq B,\ \Gamma_2$$
$$Y_\nu \in \mathrm{Def}(\Gamma'') \Rightarrow \Gamma'' = \Gamma_1,\ Y_\nu = B,\ \Gamma_2$$

Now, assume, towards a contradiction, that $\chi \in^{\simeq} \mathrm{Def}(\Gamma'')$ and $\chi = s_\beta$. In this case we have

$$\Gamma = \Gamma',\ X_\alpha = \mu X_\alpha.\forall s_\beta \leq A.A',\ s_\beta \leq A, \Gamma''$$

and

$$\Gamma'' \;=\; \Gamma_1,\, (Z_\nu{=}\mu Z_\nu.\forall s_\nu{\leq}B.B'),\, s_\nu{\leq}B,\, \Gamma_2$$

By Lemma 4.16 we have

$$\mu Z_\nu.\forall s_\nu{\leq}B.B' \in^{\simeq} \mathrm{SE}(A')$$

Now, by Lemma 4.6 we have

$$\mu X_\alpha.\forall s_\beta{\leq}A.A' \in^{\simeq} \mathrm{SE}(T)$$
$$\mu Z_\nu.\forall s_\nu{\leq}B.B' \in^{\simeq} \mathrm{SE}(T)$$

or

$$\mu X_\alpha.\forall s_\beta{\leq}A.A' \in^{\simeq} \mathrm{SE}(U)$$
$$\mu Z_\nu.\forall s_\nu{\leq}B.B' \in^{\simeq} \mathrm{SE}(U)$$

Suppose we are in the first case (the second case is similar). In this case it must be $Z = X$, because otherwise the type $T \in$LR-Types would have two different variables with the same face s, and this contradicts the uniqueness of face variables for LR-Types. Since $Z = X$, by Lemma 4.15 (similarity of bounds) we have

$$\mu Z_\nu.\forall s_\nu{\leq}B.B' \simeq \mu X_\alpha.\forall s_\beta{\leq}A.A'.$$

Hence $\mu Z_\nu.\forall s_\nu{\leq}B.B' \not\in^{\simeq} \mathrm{SE}(A')$, which contradicts Lemma 4.16.
The case $\chi = K_\beta$ is similar.□

According to Corollary 4.17, the (unique) definition in Left(Π') of each $\chi \in \mathrm{FV}(T')$, can be found by visiting Left(Π') from right to left and by stopping when a χ' s.t. $\chi \simeq \chi'$ is found. The same holds for U' with respect to Right(Π'). Hence, as stated in Theorem 4.18, by ignoring labels, we can correctly find the definition of free variables in the types compared.

Theorem 4.18 *The label-based algorithm is equivalent to a label-free one, defined as follows*

- *consider each* $\Re^{\text{alg-2}}$ *rule without labels;*
- *specify in rule* $(Id_{\leq})$ *that* $\Pi = \Pi', (t,u) \leq (\_,\_), \Pi''$ *for some* Π'' *such that, for any* v, $(t,v) \notin \mathrm{Def}(\Pi'')$ *and* $(v,u) \notin \mathrm{Def}(\Pi'')$.

Proof. By Corollary 4.17.□

A similar decomposition of Π may also be added to the premises of the (VarTrans$_{\leq}$) and unfolding rules, but this is not necessary since, by uniqueness of faces, if both $X_\alpha{=}T$ and $X_\beta{=}T'$ are in Π, then $T \simeq T'$.

However, hereafter we still consider rules $\Re^{alg\text{-}2}$ and $\Re^{\infty}$ according to their original formulation. This will make our soundness proof easier.

5 Soundness and Completeness

To prove the correctness and completeness of our algorithm we refer to the reduction trees generated by the backward application of either $\Re^{alg\text{-}2}$ or $\Re^{\infty}$ rules in verifying judgements in Start-J$^{alg\text{-}2|\infty}$. Therefore, in the first part of this section we give some definitions about such reduction trees. In Section 5.2 we give the correctness and completeness proof.

5.1 Reduction Trees

Informally, the reduction tree of a pre-judgement $\Pi \rhd_\ell T \leq U \in$ Start-J$^{\infty}$ is the tree such that: a) its root is labeled with $\Pi \rhd_\ell T \leq U$ and b) if $\Pi \rhd_\ell T \leq U$ is reduced to n pre-judgements $\Pi_i \rhd_\ell T_i \leq U_i$ by the backward application of an $\Re^{\infty}$ rule, then the children of the root are the reduction trees of the pre-judgements $\Pi_i \rhd_\ell T_i \leq U_i$. Hereafter $\mathcal{RT}_{\infty}(\Pi, T, U)$ will denote the (possibly infinite) reduction tree of $\Pi \rhd_\ell T \leq U$. Each node of a tree $\mathcal{RT}_{\infty}(\Pi, T, U)$ will contain a label ranging over the values *success* and *failure,* which respectively indicates that the pre-judgement in the node matches a rule conclusion and that it does not match any rule conclusion.

Definition 5.1 *We denote by $\mathcal{RT}|_n(\Pi, T, U)$ the reduction tree obtained by considering at most n consecutive $\Re^{\infty}$ reduction steps starting from $\Pi \rhd_\ell T \leq U$. Formally, for $n = 0$:*

$$\mathcal{RT}|_0(\Pi, T, U) = \begin{cases} (\Pi \rhd_\ell T \leq U, \mathit{success}) & \mathit{if\ the\ pre\text{-}judgement\ matches} \\ & \mathit{the\ conclusion\ of\ a}\ \Re^{\infty}\ \mathit{rule;} \\ (\Pi \rhd_\ell T \leq U, \mathit{failure}) & \mathit{otherwise.} \end{cases}$$

For $n > 0$, if $\Pi \rhd_\ell T \leq U$ is reduced to k pre-judgements $\Pi_i \rhd_\ell T_i \leq U_i$ by the backward application of an $\Re^{\infty}$ rule, then:

$$\mathcal{RT}|_n(\Pi, T, U) = \quad (\Pi \rhd_\ell T \leq U, \mathit{success})$$

$$\swarrow \quad \cdots \quad \searrow$$

$$\mathcal{RT}|_{n-1}(\Pi_1, T_1, U_1) \qquad \mathcal{RT}|_{n-1}(\Pi_k, T_k, U_k)$$

otherwise $\mathcal{RT}|_n(\Pi, T, U) = \mathcal{RT}|_{n-1}(\Pi, T, U)$.

Hereafter, for the sake of simplicity, we often ignore the position label β in $t_{\alpha|\beta}$ and $\top_\beta$ occurrences.

Definition 5.2 *For each $\Pi \rhd_\ell T \leq U \in$ Start-J^∞ we define $\mathcal{RT}_\infty(\Pi, T, U)$ as the (possibly infinite) ordered tree that satisfies the following conditions:*

- $(\rightarrow_\leq) : T = \mu X_\alpha.T' \rightarrow T'', \quad U = \mu Y_\nu.U' \rightarrow U'',$
  $\Pi' = \Pi,\ (X_\alpha = \mu X_\alpha.T' \rightarrow T'', Y_\nu = \mu Y_\nu.U' \rightarrow U'')$

  $$\Rightarrow \mathcal{RT}_\infty(\Pi, T, U) = \quad (\Pi \rhd_\ell \mu X_\alpha.T' \rightarrow T'' \leq \mu Y_\nu.U' \rightarrow U'', \mathit{success})$$
  $$\swarrow \qquad \searrow$$
  $$\mathcal{RT}_\infty(\mathrm{Swap}(\Pi'), U', T') \qquad \mathcal{RT}_\infty(\Pi', T'', U'')$$

- $(\mathit{VarTrans}_\leq)$: *for all* $X_{\theta|\delta}.\ (U \neq X_{\theta|\delta}) \quad U \neq u_{\nu|-}, \quad U \neq \top_-,$
  $\Pi = \Pi', (t_\alpha, u_\nu) \leq (T', U'), \Pi''$

  $$\Rightarrow \mathcal{RT}_\infty(\Pi, T, U) = \ (\Pi \rhd_\ell t_\alpha \leq U, \mathit{success})$$
  $$\downarrow$$
  $$\mathcal{RT}_\infty(\Pi, T', U)$$

- $(\forall_\leq) : T = \mu X_\alpha.\forall t_\alpha \leq T'.T'', \quad U = \mu Y_\nu.\forall u_\nu \leq U'.U'',$
  $\Pi' = \Pi, (X_\alpha = \mu X_\alpha.\forall t_\alpha \leq T'.T'', Y_\nu = \mu Y_\nu.\forall u_\nu \leq U'.U''),$
  $\Pi'' = \Pi', (t_\alpha, u_\nu) \leq (T', U')$

  $$\Rightarrow \mathcal{RT}_\infty(\Pi, T, U) =$$
  $$(\Pi \rhd_\ell \mu X_\alpha.\forall t_\alpha \leq T'.T'' \leq \mu Y_\nu.\forall u_\nu \leq U'.U'', \mathit{success})$$
  $$\swarrow \qquad \downarrow \qquad \searrow$$
  $$\mathcal{RT}_\infty(\Pi', T', U') \quad \mathcal{RT}_\infty(\mathrm{Swap}(\Pi'), U', T') \quad \mathcal{RT}_\infty(\Pi'', T'', U'')$$

- $(\mathit{LUnf}_\leq) : \ T = X_{\alpha|\beta}, \quad X_\alpha = T' \in \mathrm{Left}(\Pi)$

  $$\Rightarrow \mathcal{RT}_\infty(\Pi, T, U) = \qquad (\Pi \rhd_\ell X_{\alpha|\beta} \leq U, \mathit{success})$$
  $$\downarrow$$
  $$\mathcal{RT}_\infty((\Pi, X_{\alpha|\beta} \leq U), T' \uparrow \beta, U)$$

- $(RUnf_{\leq})$: $U = Y_{\nu|\eta}$, *for all* $X_{\theta|\delta}$. $(T \neq X_{\theta|\delta})$, $Y_\nu{=}U' \in \mathrm{Right}(\Pi)$

  $$\Rightarrow \mathcal{RT}_\infty(\Pi, T, U) = \quad (\Pi \triangleright_\ell T \leq Y_{\nu|\eta}, \mathit{success})$$
  $$\downarrow$$
  $$\mathcal{RT}_\infty((\Pi, T \leq Y_{\nu|\eta}), T, U' \uparrow \eta)$$

  - $(Id_{\leq})$: $T = t_\alpha$, $U = u_\nu$,
    $$\Pi = \Pi, (t_\alpha, u_\nu) \leq (T', U'), \Pi''$$

  $$\Rightarrow \mathcal{RT}_\infty(\Pi, T, U) = (\Pi \triangleright_\ell t_\alpha \leq u_\nu, \mathit{success})$$

- $(\top_{\leq})$: $U = \top$

  $$\Rightarrow\ \mathcal{RT}_\infty(\Pi, T, U) = (\Pi \triangleright_\ell T \leq \top, \mathit{success})$$

- *Failure: When none of the previous cases is satisfied, then*

  $$\mathcal{RT}_\infty(\Pi, T, U) = (\Pi \triangleright_\ell T \leq U, \mathit{fail})$$

Remark 5.3 *For a formal definition of the tree* $\mathcal{RT}_\infty(\Pi, T, U)$ *we should first define the space* $\mathbf{RT}_{fin}$ *of all possible finite trees whose nodes are labeled with pairs formed by a Start-*J^∞ *pre-judgement and a success/failure mark. This set is a metric space with respect to the usual metric on trees [AN80]. Hence, we can define the complete metric space* $\mathbf{RT}_\infty$ *obtained by the completion of* $\mathbf{RT}_{fin}$*. By definition, in this metric space, every Cauchy sequence has a limit. Hence,* $\mathcal{RT}_\infty(\Pi, T, U)$ *can be defined as the limit*

$$\mathcal{RT}_\infty(\Pi, T, U) = \lim_{n\to\infty} \mathcal{RT}|_n(\Pi, T, U)$$

The existence of this limit is guaranteed by the fact that $\{\mathcal{RT}|_n(\Pi, T, U)\}_{n\in\mathbb{N}}$ *is a Cauchy sequence.*

Note that, thanks to good formation invariance (Lemma 4.11), we do not need to consider subtrees generated by good formation checking. Also observe that, by construction, paths in $\mathcal{RT}_\infty(\Pi, T, U)$ are in one-to-one correspondence with reduction chains, starting from $\Pi \triangleright_\ell T \leq U$, produced by $\twoheadrightarrow_\infty$.

In the same way, we now define the finite reduction trees generated by the backward application of $\Re^{alg\text{-}2}$ rules.

Definition 5.4 *For each* $\Pi \triangleright_\ell T \leq U \in$ *Start-*$J^{alg|\infty}$ *we define the reduction tree* $\mathcal{RT}(\Pi, T, U)$ *as the ordered tree satisfying the following conditions (the definition differs from 5.2 only for cases concerning applications of unfolding*

or ending rules; we thus report these cases only):

- $(LEnd_{\leq})$: $T = X_{\alpha|\beta}, \quad X_{\alpha|\beta} \leq U \in_2^{\simeq} \Pi$

  $\Rightarrow \mathcal{RT}(\Pi, T, U) = (\Pi \triangleright_{\ell} X_{\alpha|\beta} \leq U, \mathit{success})$

- $(REnd_{\leq})$: *as above.*
- $(LUnf^2_{\leq})$: $T = X_{\alpha|\beta}, \quad X_{\alpha|\beta} \leq U \notin_2^{\simeq} \Pi$

  $$\Rightarrow \mathcal{RT}(\Pi, T, U) = \quad (\Pi \triangleright_{\ell} X_{\alpha|\beta} \leq U, \mathit{success})$$
  $$\downarrow$$
  $$\mathcal{RT}((\Pi, X_{\alpha|\beta} \leq U), T' \uparrow \beta, U)$$

  where $X_{\alpha} = T' \in \mathrm{Left}(\Pi)$.
- $(RUnf^2_{\leq})$: *as above.*

Observe that, thanks to termination of $\Re^{alg\text{-}2}$ rules, $\mathcal{RT}(\Pi, T, U)$ is always finite.

Since reduction trees are ordered trees where each node may have at most three children, we will represent paths on reduction trees by sequences on $\{0, 1, 2\}^*$, indicated by lowercase letters $(a, b, c \ldots)$.

Definition 5.5 *Hereafter, if a reduction tree* $\mathcal{RT}(\Pi, T, U)$ *(or* $\mathcal{RT}_{\infty}(\Pi, T, U)$*) has a node that corresponds to the path* a*, we indicate this fact with*

$$\mathcal{RT}(\Pi, T, U)(a) \downarrow$$

(resp. $\mathcal{RT}_{\infty}(\Pi, T, U)(a) \downarrow$*), and* $\mathcal{RT}(\Pi, T, U)(a)$ *(resp.* $\mathcal{RT}_{\infty}(\Pi, T, U)(a)$*) will denote the node that corresponds to the path* a*. Otherwise, we use the notation* $\mathcal{RT}(\Pi, T, U)(a) \uparrow$ *(resp.* $\mathcal{RT}_{\infty}(\Pi, T, U)(a) \uparrow$*) to indicate that no node corresponds to the path* a*.*

To simplify the notation, hereafter in reduction trees we will abbreviate the success and failure label using their initials.

5.2 Soundness and Completeness

We prove soundness by showing that, if $\mathcal{RT}((), T, U)$ is a successful tree (all nodes are labeled as successful), then it can be transformed into the successful tree $\mathcal{RT}_{\infty}((), T, U)$ by the iterated expansion of the tree $\mathcal{RT}((), T, U)$, by

eliminating all the applications of the (_End $\leq$) rules. Essentially, the idea is that if $\mathcal{RT}((),T,U)$ has a leaf of shape

$$\mathcal{RT}((),T,U)(a) = (\Pi \rhd_\ell X_{\alpha|\beta} \leq U', \mathrm{s})$$

or

$$\mathcal{RT}((),T,U)(a) = (\Pi \rhd_\ell T' \leq Y_{\nu|\eta}, \mathrm{s})$$

(such leaves will be called end-nodes) then, by considering for example the first case, we can extend $\mathcal{RT}((),T,U)$ thus obtaining a tree $\mathcal{RT}^1((),T,U)$ that is equal to $\mathcal{RT}((),T,U)$ after the leaf has been substituted with the whole finite tree

$$\begin{array}{c}(\Pi \rhd_\ell X_{\alpha|\beta} \leq U', \mathrm{s}) \\ \downarrow \\ \mathcal{RT}((\Pi, X_{\alpha|\beta} \leq U'), T', U')\end{array}$$

where $T' = \overline{T} \uparrow \beta$ and $X_\alpha = \overline{T} \in \mathrm{Left}(\Pi)$. In the same way we can obtain a tree $\mathcal{RT}^2((),T,U)$ from $\mathcal{RT}^1((),T,U)$ and, more generally, we can define a sequence

$$\{\mathcal{RT}^n((),T,U)\}_{n\in\mathbb{N}}$$

where each tree $\mathcal{RT}^n((),T,U)$ is obtained by expanding $\mathcal{RT}^{n-1}((),T,U)$ in the way just shown. To guarantee that this sequence converges to $\mathcal{RT}_\infty((),T,U)$, in each expansion step an end-node with a minimum path is considered. In the following definition we fix how this end-node, which will be called the *expansion node*, is chosen.

Definition 5.6 *For each* $() \rhd_\ell T \leq U \in$ *Start-J we define the sequence*

$$\{\mathcal{RT}^n((),T,U)\}_{n\in\mathbb{N}}$$

by induction on n:
When $n = 0$ *we consider*

$$\mathcal{RT}^0((),T,U) = \mathcal{RT}((),T,U)$$

When $n > 0$ *we distinguish the following three possible cases:*
1) (Left end-node) There is a path $a \in \{0,1,2\}^*$ *s.t. the node*

$$\mathcal{RT}^{n-1}((),T,U)(a) = (\Pi \rhd_\ell X_{\alpha|\beta} \leq U', s)$$

is a leaf of $\mathcal{RT}^{n-1}((),T,U)$ *and, for each* $a' \in \{0,1,2\}^*$ *s.t.* $\mathcal{RT}^{n-1}((),T,U)(a')$ *is an end-node, then* $|a| \leq |a'|$*. In this case we define* $\mathcal{RT}^n((),T,U)$ *as the tree*

obtained by replacing the leaf $\mathcal{RT}^{n-1}((),T,U)(a)$ *with the whole tree:*

$$\begin{array}{c}(\Pi \rhd_\ell X_{\alpha|\beta} \leq U', s)\\ \downarrow\\ \mathcal{RT}((\Pi, X_{\alpha|\beta} \leq U'), T', U')\end{array}$$

where

$T' = \overline{T} \uparrow \beta$ *and* $X_\alpha = \overline{T} \in \text{Left}(\Pi)$.

2) (Right end-node) There is an end-node $\mathcal{RT}^{n-1}((),T,U)(a)$ *with a minimal length path a and with shape*

$(\Pi \rhd_\ell T' \leq Y_{\nu|\eta}, s)$.

In this case we define $\mathcal{RT}^n((),T,U)$ *as in the previous case, i.e. by replacing* $\mathcal{RT}^{n-1}((),T,U)(a)$ *with the tree*

$$\begin{array}{c}(\Pi \rhd_\ell T' \leq Y_{\nu|\eta}, s)\\ \downarrow\\ \mathcal{RT}((\Pi, T' \leq Y_{\nu|\eta}), T', U')\end{array}$$

where

$U' = \overline{U} \uparrow \eta$ *and* $Y_\nu = \overline{U} \in \text{Right}(\Pi)$

3)When $\mathcal{RT}^{n-1}((),T,U)$ *has no end-node, we consider:*

$\mathcal{RT}^n((),T,U) = \mathcal{RT}^{n-1}((),T,U)$

We assume that case 1 has priority over case 2. Moreover, we assume that when $\mathcal{RT}^{n-1}((),T,U)$ *has more than one leaf that satisfies case 1, the leftmost leaf is considered; the same applies when case 2 is considered.*

According to this definition, for each $()\rhd_\ell T \leq U \in$ Start-J we have defined a sequence

$\{\mathcal{RT}^n((),T,U)\}_{n\in\mathbb{N}}$

The sequence $\{\mathcal{RT}^n((),T,U)\}_{n\in\mathbb{N}}$ is clearly a Cauchy sequence in the complete metric space $\mathbf{RT}_\infty$, hence it has a limit. Moreover, the way we choose the expansion node ensures that the limit is $\mathcal{RT}_\infty((),T,U)$:

Property 5.7 *For each* $() \rhd_\ell T \leq U \in$ *Start-J,*

$\lim_{n\to\infty} \mathcal{RT}^n((),T,U) = \mathcal{RT}_\infty((),T,U)$

Proof. Any end-node in a $\mathcal{RT}^n((), T, U)$ is expanded away before step $k^{n+1}+1$, where k is the number of expansion nodes of $\mathcal{RT}^0((), T, U)$.□

Lemma 5.8 *For each* $() \rhd_\ell T \leq U \in$ *Start-J,* $\mathcal{RT}_\infty((), T, U)$ *is successful if and only if each* $\mathcal{RT}^n((), T, U)$ *is successful.*

Proof. Observe that each failure in the limit $\mathcal{RT}_\infty((), T, U)$ is a failure in some $\mathcal{RT}^n((), T, U)$ and vice versa, and apply Lemma 5.7.□

In the next part we will need the following definition and lemmas.

Definition 5.9 *We say that the tree* $\mathcal{RT}^n((), T, U)$ *is* expandable *if it has at least one end-node. In this case, according to the previous definition, the node that is expanded to transform* $\mathcal{RT}^n((), T, U)$ *into* $\mathcal{RT}^{n+1}((), T, U)$*, will be called the* expansion node*.*

Lemma 5.10 (Forward node equality) *For each* $() \rhd_\ell T \leq U \in$ *Start-J,* $k, i \in \mathbb{N}$ *and* $a \in \{0, 1, 2\}^*$:

$$\mathcal{RT}^k((), T, U)(a) \downarrow \Rightarrow \mathcal{RT}^k((), T, U)(a) = \mathcal{RT}^{k+i}((), T, U)(a)$$

Proof. By induction on i.□

Lemma 5.11 (Backward node equality) *If* $() \rhd_\ell T \leq U \in$ *Start-J, and the tree* $\mathcal{RT}^k((), T, U)$ *is expandable by the expansion node* $\mathcal{RT}^k((), T, U)(a)$*, then for each* $i \geq 0$ *and* a' *s.t.*

$$\mathcal{RT}^{k+i}((), T, U)(a') \downarrow$$
$$|a'| \leq |a|$$

we have

$$\mathcal{RT}^k((), T, U)(a') \downarrow$$
$$\mathcal{RT}^k((), T, U)(a') = \mathcal{RT}^{k+i}((), T, U)(a')$$

Proof. By induction on i and by minimality of a.□

Lemma 5.8 implies that soundness and completeness can be established by proving that, for each $() \rhd_\ell T \leq U \in$ Start-J, $\mathcal{RT}((), T, U)$ is a successful tree if and only if each $\mathcal{RT}^n((), T, U)$ is successful (i.e. $\mathcal{RT}_\infty((), T, U)$ is successful). While the *if* implication is trivially true since

$$\mathcal{RT}((), T, U) =_{\text{def}} \mathcal{RT}^0((), T, U),$$

we will prove the *only if* direction by induction on n, i.e. by proving that if $\mathcal{RT}^n((), T, U)$ is a successful tree then so is $\mathcal{RT}^{n+1}((), T, U)$.

We know that $\mathcal{RT}^{n+1}((), T, U)$ differs from $\mathcal{RT}^{n}((), T, U)$ only with respect to a subtree that is rooted at the expansion node of the latter. We will see that the key of the problem of showing that this subtree is successful, is to prove that each new node $\overline{\Pi} \rhd_\ell t_\nu \leq u_\eta$ created in the subtree (two type variables are compared) is the root of a successful tree. To this end we will prove that similar successful subtrees rooted at the nodes $\overline{\overline{\Pi}} \rhd_\ell t_\tau \leq u_\upsilon$ exist in $\mathcal{RT}^{n}((), T, U)$ and that this implies that the new subtree rooted at $\overline{\Pi} \rhd_\ell t_\nu \leq u_\eta$ is successful as well. Observe that this is not true if we consider $\Re^{alg\text{-}1}$ instead of $\Re^{alg\text{-}2}$ rules. Indeed, in the counterexample of Section 4.2, although the node $\overline{\overline{\Pi}} \rhd_\ell t_\tau \leq u_\upsilon$ is a successful leaf, we have a new node $\overline{\Pi} \rhd_\ell t_\nu \leq u_\eta$ which is reduced, by rule ($\text{VarTrans}_{\leq}$), into a failure node.

Below, we outline properties for this kind of (Π, t_α, u_β)-rooted subtrees of a successful tree. These properties will be used to prove the soundness of the subtyping algorithm defined by $\Re^{alg\text{-}2}$ rules.

Lemma 5.12 *If* $() \rhd_\ell T \leq U \in$ *Start-J,* $\mathcal{RT}^{n}((), T, U)$ *is a successful tree, and*

$$\mathcal{RT}^{n}((), T, U)(a) = (\overline{\Pi} \rhd_\ell \mu X_\alpha.\forall t_\alpha \leq t'_{\alpha'}.T' \ \leq \ \mu Y_\beta.\forall u_\beta \leq u'_{\beta'}.U', s)$$

then $(t'_{\alpha'}, u'_{\beta'}) \in \text{Def}(\overline{\Pi})$.

Proof. By hypothesis we have:

$$\mathcal{RT}^{n}((), T, U)(a.0) =$$
$$(\overline{\Pi}, (X_\alpha = \mu X_\alpha.\forall t_\alpha \leq t'_{\alpha'}.T', Y_\beta = \mu Y_\beta.\forall u_\beta \leq u'_{\beta'}.U') \rhd_\ell t'_{\alpha'} \leq u'_{\beta'}, \text{s})$$
$$\mathcal{RT}^{n}((), T, U)(a.1) =$$
$$(\text{Swap}(\overline{\Pi}, (X_\alpha = \mu X_\alpha.\forall t_\alpha \leq t'_{\alpha'}.T', Y_\beta = \mu Y_\beta.\forall u_\beta \leq u'_{\beta'}.U')) \rhd_\ell u'_{\beta'} \leq t'_{\alpha'}, \text{s})$$

If we suppose that one of these two nodes is not a leaf, then it is easy to prove that the other one is a failure node. So, both nodes are leaves of the reduction tree, hence $(t'_{\alpha'}, u'_{\beta'}) \in \text{Def}(\overline{\Pi})$.□

Corollary 5.13 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^{n}((), T, U)$ *is a successful tree such that*

$$\mathcal{RT}^{n}((), T, U)(a) = (\overline{\Pi} \rhd_\ell \overline{T} \leq \overline{U}, s),$$

where $\overline{\Pi} = (\overline{\Pi}_1, (t_\alpha, u_\beta) \leq (t'_{\alpha'}, u'_{\beta'}), \overline{\Pi}_2)$, *then we have*

$$\overline{\Pi}_1 = \overline{\Pi}'_1, (t'_{\alpha'}, u'_{\beta'}) \leq (A, B), \overline{\Pi}''_1$$

Proof. By induction on the length of $\overline{\Pi}_2$ and by the previous lemma.□

Lemma 5.14 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful tree such that*

$$\mathcal{RT}^n((), T, U)(a) = (\overline{\Pi} \rhd_\ell t_\alpha \leq u_\beta, s)$$

then either $(t_\alpha, u_\beta) \in \mathrm{Def}(\overline{\Pi})$ *and* $\mathcal{RT}^n((), T, U)(a)$ *is a leaf proved by* $(Id_\leq)$*, or* $(t_\alpha, u_\beta) \notin \mathrm{Def}(\overline{\Pi})$ *and:*

- *1. the subtree rooted at* $\mathcal{RT}^n((), T, U)(a)$ *is generated by* $k \geq 1$ *applications of* $(VarTrans_\leq)$ *and has the following shape:*

$$\begin{array}{c} (\overline{\Pi} \rhd_\ell t_\alpha \leq u_\beta, s) \\ \downarrow \\ (\overline{\Pi} \rhd_\ell t^1_{\alpha^1} \leq u_\beta, s) \\ \downarrow \\ \vdots \\ (\overline{\Pi} \rhd_\ell t^{k-1}_{\alpha^{k-1}} \leq u_\beta, s) \\ \downarrow \\ (\overline{\Pi} \rhd_\ell t^k_{\alpha^k} \leq u_\beta, s) \end{array}$$

- *2.* $\overline{\Pi}$ *has the following structure:*

$$\begin{array}{l} \overline{\Pi} = \overline{\Pi}_{k+1}, (t^k_{\alpha^k}, u_\beta) \leq (T', U'), \\ \quad \overline{\Pi}_k, (t^{k-1}_{\alpha^{k-1}}, u^{k-1}_{\beta^{k-1}}) \leq (t^k_{\alpha^k}, u_\beta), \\ \quad \overline{\Pi}_2, (t^1_{\alpha^1}, u^1_{\beta^1}) \leq (t^2_{\alpha^2}, u^2_{\beta^2}), \\ \quad \overline{\Pi}_1, (t_\alpha, u^0_{\beta^0}) \leq (t^1_{\alpha^1}, u^1_{\beta^1}), \overline{\Pi}_0 \end{array}$$

  where $u^i \neq u$ *for* $i = 0 \ldots k-1$.

Proof. Since $\mathcal{RT}^n((), T, U)$ is successful, $\overline{\Pi} \rhd_\ell t_\alpha \leq u_\beta$ is provable. If it is proved by $(\mathrm{Id}_\leq)$ then $(t_\alpha, u_\beta) \in \mathrm{Def}(\overline{\Pi})$; otherwise $(t_\alpha, u_\beta) \notin \mathrm{Def}(\overline{\Pi})$. In this case we proceed by induction on k, the depth of the tree rooted at $\mathcal{RT}^n((), T, U)(a)$, and prove 1 and 2 together.

We first note that since $(t_\alpha, u_\beta) \notin \mathrm{Def}(\overline{\Pi})$ then :

$$\overline{\Pi} = \overline{\Pi}',\ (t_\alpha, u^0_{\beta^0}) \leq (\overline{T}, \overline{U}), \overline{\Pi}_0 \quad \wedge \quad u_\beta \neq u^0_{\beta^0}$$

and the pre-judgement $\overline{\Pi} \rhd_\ell t_\alpha \leq u_\beta$ is reduced to $\overline{\Pi} \rhd_\ell \overline{T} \leq u_\beta$ by $(\mathrm{VarTrans}_\leq)$. Since $\overline{\Pi} \rhd_\ell \overline{T} \leq u_\beta$ is provable, $\overline{T}$ must be a type variable $t^1_{\alpha^1}$, that is $\overline{T} = t^1_{\alpha^1}$ (in the other case this pre-judgement would correspond

to a failure node in the reduction tree $\mathcal{RT}^n((), T, U))$. Moreover, by the $(\forall_\leq)$ rule, which has inserted $(t_\alpha, u^0_{\beta^0}) \leq (\overline{T}, \overline{U})$ in $\overline{\Pi}$ by requiring $\overline{\Pi}' \vdash_\ell \overline{T} = \overline{U}$ with $\overline{T} = t^1_{\alpha^1}$, also $\overline{U}$ must be a type variable $u^1_{\beta^1}$. Hence we actually have

$$\overline{\Pi} = \overline{\Pi}',\ (t_\alpha, u^0_{\beta^0}) \leq (t^1_{\alpha^1}, u^1_{\beta^1}), \overline{\Pi}_0$$

and, by Corollary 5.13, we have

$$\overline{\Pi}' = \overline{\Pi}'', (t^1_{\alpha^1}, u^1_{\beta^1}) \leq (\overline{\overline{T}}, \overline{\overline{U}}), \overline{\Pi}^1$$

These properties imply that $\overline{\Pi} \rhd_\ell t_\alpha \leq u_\beta$ is reduced to $\overline{\Pi} \rhd_\ell t^1_{\alpha^1} \leq u_\beta$ by (VarTrans$_\leq$) and that $u^1_{\beta^1} = u_\beta$ in the case that (Id$_\leq$) application is possible for $\overline{\Pi} \rhd_\ell t^1_{\alpha^1} \leq u_\beta$ (case $k = 1$). Otherwise (case $k \geq 2$), we have $u^1_{\beta^1} \neq u_\beta$ and, since $u_\beta \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}''))$, by 4.17 we also have $u^1 \neq u$. Then it is sufficient to apply induction to

$$\mathcal{RT}^n((), T, U)(a.0) = (\overline{\Pi} \rhd_\ell t^1_{\alpha^1} \leq u_\beta, \mathrm{s})$$

so as to obtain the indicated shape of the sub-tree $\mathcal{RT}^n((), T, U)(a)$ given in 1, and the structure of $\overline{\Pi}$ together with inequalities $u^i \neq u$ given in 2.□

The following lemma is needed to justify Definition 5.16.

Lemma 5.15 *For each* $() \rhd_\ell T \leq U \in$ *Start-J and* $\Pi \rhd_\ell T' \leq U'$ *such that*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi \rhd_\ell T' \leq U', \_)$$

if $\Gamma = \mathrm{Left}(\Pi)$ *and* $\Gamma = \Gamma', t_\alpha \leq s_{\beta|\eta}, \Gamma''$, *then* $s_\beta \in \mathrm{Def}(\Gamma')$ *and* $s_\beta \notin \mathrm{Def}(\Gamma'')$. *The same holds for* $\Gamma = \mathrm{Right}(\Pi)$.

Proof. We prove the case $\Gamma = \mathrm{Left}(\Pi)$, the other one is the same. Assumptions $\Gamma = \mathrm{Left}(\Pi)$ and $\Gamma = \Gamma', t_\alpha \leq s_{\beta|\eta}, \Gamma''$ mean that

$$\Pi = \Pi', (t_\alpha, u_\delta) \leq (s_{\beta|\eta}, A), \Pi''$$
$$\Gamma' = \mathrm{Left}(\Pi')$$
$$\Gamma'' = \mathrm{Left}(\Pi'')$$

By Corollary 4.11, we have

$$\mathrm{FV}(s_{\beta|\eta}) = \{s_\beta\} \subseteq \mathrm{Def}(\mathrm{Left}(\Pi')) = \mathrm{Def}(\Gamma')$$

which entails $s_\beta \in \mathrm{Def}(\Gamma')$. Therefore, if we assume $s_\beta \in \mathrm{Def}(\Gamma'')$, Lemma 4.9 would be contradicted, since we would have s_β defined twice in $\Gamma = \mathrm{Left}(\Pi) = (\Gamma', t_\alpha \leq s_{\beta|\eta}, \Gamma'')$ and the label β occurring twice in the sequence of labels of type variables defined in $\mathrm{Left}(\Pi)$. □

Definition 5.16 *For each* $() \rhd_\ell T \leq U \in$ *Start-J and* $\Pi \rhd_\ell T' \leq U'$ *such that*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi \rhd_\ell T' \leq U', \_)$$

if $\Gamma = \text{Left}(\Pi)$ *(or* $\Gamma = \text{Right}(\Pi)$*) and* $t_\alpha \in \text{Def}(\Gamma)$*, we define* $\text{Bounds}(t_\alpha,\ \Gamma)$*, the set of direct and indirect type-variable bounds of* t_α *in* Γ*, as follows*

$$\text{Bounds}(t_\alpha,\ \Gamma) = \begin{cases} \{t_\alpha\} \cup \text{Bounds}(s_\beta,\ \Gamma') & \text{if } \Gamma = \Gamma',\ t_\alpha \leq s_{\beta|\eta},\ \Gamma'' \text{ for some type variable } s_\beta \\ \{t_\alpha\} & \text{otherwise} \end{cases}$$

In the following we denote with $\#\text{Bounds}(t_\alpha,\ \Gamma)$ the number $|\text{Bounds}(t_\alpha,\ \Gamma)|$.

Lemma 5.17 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful tree such that*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi' \rhd_\ell T' \leq U', s)$$

and $(t_\alpha, u_\beta) \in \text{Def}(\Pi')$*, then*

$$\#\text{Bounds}(t_\alpha,\ \text{Left}(\Pi')) = \#\text{Bounds}(u_\beta,\ \text{Right}(\Pi'))$$

moreover

$$\forall\ t'_{\alpha'} \in \text{Bounds}(t_\alpha,\ \text{Left}(\Pi')).\ \exists\ u'_{\beta'} \in \text{Bounds}(u_\beta,\ \text{Right}(\Pi'))\ \textit{s.t.}\ (t'_{\alpha'}, u'_{\beta'}) \in \text{Def}(\Pi')$$

and vice versa.

Proof. Easy induction on $\#\text{Bounds}(t_\alpha,\ \text{Left}(\Pi'))$ (use Lemma 5.14).□

Lemma 5.18 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful tree such that*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi \rhd_\ell t_\alpha \leq u_\beta, s)$$

and $u'_{\beta'}$ *is the variable unified in* Π *with* t_α*, i.e.* $(t_\alpha, u'_{\beta'}) \in \text{Def}(\Pi)$*, then* $u_\beta \in \text{Bounds}(u'_{\beta'},\ \text{Right}(\Pi))$*. Moreover, if* $u_\beta \neq u'_{\beta'}$*, then*

$$\#\text{Bounds}(u_\beta,\ \text{Right}(\Pi)) < \#\text{Bounds}(u'_{\beta'},\ \text{Right}(\Pi))$$

Proof. Easy induction on the depth of the subtree rooted at $\mathcal{RT}^n((), T, U)(a)$ — remember that this subtree has the shape illustrated in Lemma 5.14.□

Lemma 5.19 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful tree with two nodes*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi' \rhd_\ell T' \leq U', s)$$

and

$$\mathcal{RT}^n((), T, U)(b) = (\Pi'' \rhd_\ell T'' \leq U'', s)$$

such that $t_\alpha \in \mathrm{Def}(\Gamma')$, *with* $\Gamma' = \mathrm{Left}(\Pi')$ *or* $\Gamma' = \mathrm{Right}(\Pi')$, *and* $t_{\alpha'} \in \mathrm{Def}(\Gamma'')$, *with* $\Gamma'' = \mathrm{Left}(\Pi'')$ *or* $\Gamma'' = \mathrm{Right}(\Pi'')$, *then*

$$\#\mathrm{Bounds}(t_\alpha,\ \Gamma') = \#\mathrm{Bounds}(t_{\alpha'},\ \Gamma'')$$

Proof. By induction on $\#\mathrm{Bounds}(t_\alpha,\ \Gamma')$, uniqueness of face variables in $() \rhd_\ell T \leq U$ and similarity of bounds (Lemma 4.15).□

Lemma 5.20 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful tree such that*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi \rhd_\ell t_\alpha \leq u_\beta, s)$$

and

$$\mathcal{RT}^n((), T, U)(b) = (\overline{\Pi} \rhd_\ell t_\tau \leq u_\eta, s)$$

then

$$\begin{array}{c} (\Pi \rhd_\ell t_\alpha \leq u_\beta, s) \\ \downarrow \\ (\Pi \rhd_\ell t^1_{\alpha^1} \leq u_\beta, s) \end{array}$$

if and only if

$$\begin{array}{c} (\overline{\Pi} \rhd_\ell t_\nu \leq u_\eta, s) \\ \downarrow \\ (\overline{\Pi} \rhd_\ell t^1_{\nu^1} \leq u_\eta, s) \end{array}$$

Proof. It is enough to prove that $(t_\alpha, u_\beta) \notin \mathrm{Def}(\Pi)$ iff $(t_\nu, u_\eta) \notin \mathrm{Def}(\overline{\Pi})$. Then the claim easily follows by uniqueness of the bound of type variables and by the assumption stating that $\mathcal{RT}^n((), T, U)$ is successful.

Suppose that $(t_\alpha, u_\beta) \notin \mathrm{Def}(\Pi)$, then by assumptions and by Lemma 5.14 we have that

$$\overline{\Pi} = \overline{\Pi}_1, (t_\alpha, u^0_{\beta^0}) \leq (t^1_{\alpha^1}, u^1_{\beta^1}), \overline{\Pi}_0$$

and $u_\beta \neq u^0_{\beta^0}$. Hence, by Lemma 5.18 and by Lemma 5.17, we have

$$\#\mathrm{Bounds}(u_\beta,\ \mathrm{Right}(\overline{\Pi})) < \#\mathrm{Bounds}(u^0_{\beta^0},\ \mathrm{Right}(\overline{\Pi}))$$

and $\#\text{Bounds}(u^0_{\beta^0},\ \text{Right}(\overline{\Pi})) = \#\text{Bounds}(t_\alpha,\ \text{Left}(\overline{\Pi}))$.

By this and by Lemma 5.19, we have that

$$\#\text{Bounds}(u_\eta,\ \text{Right}(\overline{\Pi})) < \#\text{Bounds}(t_\nu,\ \text{Left}(\overline{\Pi}))$$

So, it cannot be $(t_\nu, u_\eta) \in \text{Def}(\overline{\Pi})$, because in this case by Lemma 5.17 we would have

$$\#\text{Bounds}(u_\eta,\ \text{Right}(\overline{\Pi})) = \#\text{Bounds}(t_\nu,\ \text{Left}(\overline{\Pi}))$$

which contradicts the previous inequality.□

Lemma 5.21 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful tree such that*

$$\mathcal{RT}^n((), T, U)(a) = (\Pi, \Pi' \rhd_\ell t_\alpha \leq u_\beta, s)$$
$$\mathcal{RT}^n((), T, U)(b) = (\Pi, \Pi'' \rhd_\ell t_\tau \leq u_\eta, s)$$

and

$$t_\alpha,\ t_\tau\ \in \text{Def}(\text{Left}(\Pi))$$
$$u_\beta,\ u_\eta \in \text{Def}(\text{Right}(\Pi))$$

then $t_\alpha = t_\tau$ *and* $u_\beta = u_\eta$.

Proof. We only prove that $t_\alpha = t_\tau$, the proof of the second equality is analogous.
Given $\Gamma = \text{Left}(\Pi)$, by hypothesis we have

$$\Gamma = \Gamma', t_\alpha \leq T', \Gamma''$$

hence by Corollary 4.17

$$\nexists \rho \text{ s.t. } t_\rho \in \text{Def}(\Gamma'') \quad (1)$$

Now, towards a contradiction, suppose that $t_\alpha \neq t_\tau$, then it must be either $t_\tau \in \text{Def}(\Gamma')$ or $t_\tau \in \text{Def}(\Gamma'')$. The second case contradicts (1). So it may only be $t_\tau \in \text{Def}(\Gamma')$, that is

$$\Gamma = \Gamma'_1, t_\tau \leq T'', \Gamma'_2, t_\alpha \leq T', \Gamma''$$

but, by Corollary 4.17, we have

$$\nexists \rho \text{ s.t. } t_\rho \in \text{Def}(\Gamma'_2, t_\alpha \leq T', \Gamma'')$$

which is absurd.□

Notation 5.22 *Hereafter,* Swap(Π) *will be abbreviated to* Π^{-1}.

Definition 5.23 *We extend the relation* $\simeq$*, which is defined over LR-Types, to environments* Γ *and bi-environments* Π *as follows:*
environments Γ :

$$\begin{aligned}
&() && \simeq () \\
&\Gamma, t_\alpha \leq T && \simeq \overline{\Gamma}, t_{\alpha'} \leq \overline{T} && \Leftrightarrow \Gamma \simeq \overline{\Gamma},\ T \simeq \overline{T} \\
&\Gamma, X_\alpha = T && \simeq \overline{\Gamma}, X_{\alpha'} = \overline{T} && \Leftrightarrow \Gamma \simeq \overline{\Gamma},\ T \simeq \overline{T} \\
&\Gamma, T && \simeq \overline{\Gamma}, \overline{T} && \Leftrightarrow \Gamma \simeq \overline{\Gamma},\ T \simeq \overline{T}
\end{aligned}$$

bi-environments Π :

$$\Pi \simeq \overline{\Pi} \ \Leftrightarrow\ \mathrm{Left}(\Pi) \simeq \mathrm{Left}(\overline{\Pi}),\ \mathrm{Right}(\Pi) \simeq \mathrm{Right}(\overline{\Pi})$$

The next two lemmas highlight interesting properties of a generic successful and expandable reduction tree $\mathcal{RT}^n((), T, U)$ where $() \rhd_\ell T \leq U \in$Start-J.

Lemma 5.24 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful and expandable tree whose expansion node is*

$$\mathcal{RT}^n((), T, U)(a) = (\overline{\Pi} \rhd_\ell X_{\alpha|\beta} \leq U', s),$$

then $\exists\, a_1, a_2, a_3$ *such that the following properties hold:*

(1) $a = a_1.a_2.a_3$ *and*

$$\begin{aligned}
\mathcal{RT}^n((), T, U)(a_1) &= (\overline{\Pi}^1 \rhd_\ell X_{\alpha^1|\beta^1} \leq U^1, s) \\
\mathcal{RT}^n((), T, U)(a_1.a_2) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2 \rhd_\ell X_{\alpha^2|\beta^2} \leq U^2, s) \\
\mathcal{RT}^n((), T, U)(a_1.a_2.a_3) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \\
&\qquad\qquad \overline{\Pi}^3 \rhd_\ell X_{\alpha|\beta} \leq U', s) \\
&= (\overline{\Pi} \rhd_\ell X_{\alpha|\beta} \leq U', s)
\end{aligned}$$

where

$$\begin{aligned}
&U^1 \simeq U^2 \simeq U' \\
&X_{\alpha|\beta} \leq U' \not\in_1^{\simeq} \overline{\Pi}^1 \\
&X_{\alpha|\beta} \leq U' \not\in_1^{\simeq} \overline{\Pi}^3
\end{aligned}$$

(2) $\forall b \in \{0,1,2\}^*$ *such that* $\mathcal{RT}^n((),T,U)(a_1.a_2.b)\downarrow$ *we have that*
(a) either

$$\mathcal{RT}^n((),T,U)(a_1.a_2.b) = (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\Pi}^* \rhd_\ell A \leq B, s)$$

and

$$\mathcal{RT}^n((),T,U)(a_1.b) = (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^{**} \rhd_\ell \overline{A} \leq \overline{B}, s)$$

or

$$\mathcal{RT}^n((),T,U)(a_1.a_2.b) = ((\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2)^{-1}, \overline{\Pi}^* \rhd_\ell A \leq B, s)$$

and

$$\mathcal{RT}^n((),T,U)(a_1.b) = ((\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1)^{-1}, \overline{\Pi}^{**} \rhd_\ell \overline{A} \leq \overline{B}, s),$$

where $\overline{\Pi}^* \simeq \overline{\Pi}^{**}$, $A \simeq \overline{A}$, $B \simeq \overline{B}$.
(b) in particular:

$$\mathcal{RT}^n((),T,U)(a_1.a_3) = (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^3 \rhd_\ell X_{\alpha^4|\beta^4} \leq U^4, s)$$

where $\overline{\overline{\Pi}}^3 \simeq \overline{\Pi}^3$, $U^4 \simeq U'$. *Moreover, by the minimality of* a *and by* $a_2 \neq nil$, $\mathcal{RT}^n((),T,U)(a_1.a_3)$ *is not a leaf.*

Proof.

(1) By hypothesis we have

$$\mathcal{RT}^n((),T,U)(a) = (\overline{\Pi} \rhd_\ell X_{\alpha|\beta} \leq U', \mathrm{s})$$

and

$$X_{\alpha|\beta} \leq U' \in_2^{\simeq} \overline{\Pi}$$

Therefore in $\overline{\Pi}$ we have at least two occurrences of pairs $X_{-|-} \leq \overline{U}$ where $U' \simeq \overline{U}$. Now, if we consider

$$X_{\alpha^1|\beta^1} \leq U^1$$
$$X_{\alpha^2|\beta^2} \leq U^2$$

as the first and the last of such a pairs, then

$$\overline{\Pi} = \overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\Pi}^3$$

and

$$U^1 \simeq U^2 \simeq U'$$
$$X_{\alpha|\beta} \leq U' \not\in \overline{\Pi}^1$$
$$X_{\alpha|\beta} \leq U' \not\in \overline{\Pi}^3$$

Moreover, due to the stop condition that $\Re^{alg\text{-}2}$ adopts, and since:

$$\begin{aligned}(\overline{\Pi}^1 \rhd_\ell X_{\alpha^1|\beta^1} \leq U^1) &\twoheadrightarrow_{\text{alg}|\infty} (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2 \rhd_\ell X_{\alpha^2|\beta^2} \leq U^2) \\ &\twoheadrightarrow_{\text{alg}|\infty} (\overline{\Pi} \rhd_\ell X_{\alpha|\beta} \leq U')\end{aligned}$$

then $\exists\, a_1, a_2, a_3$ s.t. $a = a_1.a_2.a_3$ and

$$\begin{aligned}\mathcal{RT}^n((), T, U)(a_1) &= (\overline{\Pi}^1 \rhd_\ell X_{\alpha^1|\beta^1} \leq U^1, \text{s}) \\ \mathcal{RT}^n((), T, U)(a_1.a_2) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2 \rhd_\ell X_{\alpha^2|\beta^2} \leq U^2, \text{s}) \\ \mathcal{RT}^n((), T, U)(a_1.a_2.a_3) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \\ &\qquad \overline{\Pi}^3 \rhd_\ell X_{\alpha|\beta} \leq U', \text{s}) \\ &= (\overline{\Pi} \rhd_\ell X_{\alpha|\beta} \leq U', \text{s}).\end{aligned}$$

(2) By induction on $|b|$. If $|b| = 0$ then $b = nil$ and the conclusion follows by (1). If $|b| = n > 0$ then $b = b'.\bar{b}$ where $\bar{b} \in \{0, 1, 2\}$. By $b' \prec_p b$ and $\mathcal{RT}^n((), T, U)(a_1.a_2.b) \downarrow$ we have that $\mathcal{RT}^n((), T, U)(a_1.a_2.b') \downarrow$, so by induction we have:

$$\begin{aligned}&\mathcal{RT}^n((), T, U)(a_1.a_2.b') = \\ &\qquad (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\Pi}^* \rhd_\ell A \leq B, \text{s}) \qquad \text{(a1)}\end{aligned}$$

or

$$\begin{aligned}&\mathcal{RT}^n((), T, U)(a_1.a_2.b') = \\ &\qquad ((\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1 \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2,)^{-1}, \overline{\Pi}^* \rhd_\ell A \leq B, \text{s}) \text{ (a2)}.\end{aligned}$$

We consider here the first case, the second case is similar.
By induction hypothesis we have

$$\mathcal{RT}^n((), T, U)(a_1.b') = (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^{**} \rhd_\ell \overline{A} \leq \overline{B}, \text{s})$$

where

$$\overline{\Pi}^* \simeq \overline{\Pi}^{**}$$
$$A \simeq \overline{A}$$
$$B \simeq \overline{B}.$$

The proof continues by cases on the shape of A and B, and then by cases on the rule applied to expand the node $\mathcal{RT}^n((), T, U)(a_1.a_2.b')$. The most interesting cases are ($\text{LUnf}^2_{\leq}$), ($\rightarrow_{\leq}$) and ($\text{VarTrans}_{\leq}$).
In the first one, we have that either this node has been an expansion node for $\mathcal{RT}^k((), T, U)$ with $k < n$ (case i) or that for this node the stop condition of the subtyping algorithm is not satisfied (case ii). In case (i), by minimality of $|a_1.a_2.b'|$ in $\mathcal{RT}^k((), T, U)$ and by $|a_1.b'| < |a_1.a_2.b'|$ we have that $\mathcal{RT}^k((), T, U)(a_1.b')$ is not a leaf, and by Lemma 5.10 $\mathcal{RT}^n((), T, U)(a_1.b')$ is not a leaf either. In case (ii), by $\overline{\Pi}^* \simeq \overline{\Pi}^{**}$ in the induction hypothesis, we also have that for $\mathcal{RT}^n((), T, U)(a_1.b')$ the stop condition is not satisfied, hence it cannot be a leaf. Now, the thesis follows by applying Lemma 4.15 (similarity of bounds).
To prove case ($\rightarrow_{\leq}$) it is sufficient to consider the bi-environment swapping and the similarity between A and $\overline{A}$ and between B and $\overline{B}$.
Finally, if we are in the ($\text{VarTrans}_{\leq}$) case (this means that A and $\overline{A}$ are similar type variables), we further distinguish these two sub cases: (i) both B and $\overline{B}$ are not type variables, (ii) both B and $\overline{B}$ are similar type variables. In case (i) the thesis follows by Lemma 4.15 (similarity of bounds). For case (ii) it is sufficient to apply Lemma 5.20.□

The next lemma is just the same as the previous one, where the left and right hand sides are swapped.

Lemma 5.25 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful and expandable tree whose expansion node is*

$$\mathcal{RT}^n((), T, U)(a) = (\overline{\Pi} \rhd_\ell T' \leq Y_{\alpha|\beta}, s)$$

then $\exists\, a_1, a_2, a_3$ *such that the following three properties hold:*

(1) $a = a_1.a_2.a_3$ *where*

$$\begin{aligned}
\mathcal{RT}^n((), T, U)(a_1) &= (\overline{\Pi}^1 \rhd_\ell T^1 \leq Y_{\alpha^1|\beta^1}, s)\\
\mathcal{RT}^n((), T, U)(a_1.a_2) &= (\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1}, \overline{\Pi}^2 \rhd_\ell T^2 \leq Y_{\alpha^2|\beta^2}, s)\\
\mathcal{RT}^n((), T, U)(a_1.a_2.a_3) &= (\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1}, \overline{\Pi}^2, T^2 \leq Y_{\alpha^2|\beta^2},\\
&\qquad\qquad \overline{\Pi}^3 \rhd_\ell T' \leq Y_{\alpha|\beta}, s)\\
&= (\overline{\Pi} \rhd_\ell T' \leq Y_{\alpha|\beta}, s)
\end{aligned}$$

and

$$\begin{aligned}
&T^1 \simeq T^2 \simeq T'\\
&T' \leq Y_{\alpha|\beta} \not\in^{\simeq}_1 \overline{\Pi}^1\\
&T' \leq Y_{\alpha|\beta} \not\in^{\simeq}_1 \overline{\Pi}^3
\end{aligned}$$

(2) $\forall b \in \{0,1,2\}^*$ *such that* $\mathcal{RT}^n((),T,U)(a_1.a_2.b)\downarrow$ *then:*
(a) either

$$\mathcal{RT}^n((),T,U)(a_1.a_2.b) =$$
$$(\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1}, \overline{\Pi}^2, T^2 \leq Y_{\alpha^2|\beta^2}, \overline{\Pi}^* \rhd_\ell A \leq B, s)$$
$$\textit{and}\ \ \mathcal{RT}^n((),T,U)(a_1.b) = (\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1}, \overline{\Pi}^{**} \rhd_\ell \overline{A} \leq \overline{B}, s)$$

or

$$\mathcal{RT}^n((),T,U)(a_1.a_2.b) =$$
$$((\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1}, \overline{\Pi}^2, T^2 \leq Y_{\alpha^2|\beta^2})^{-1}, \overline{\Pi}^* \rhd_\ell A \leq B, s)$$
and
$$\mathcal{RT}^n((),T,U)(a_1.b) = ((\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1})^{-1}, \overline{\Pi}^{**} \rhd_\ell \overline{A} \leq \overline{B}, s)$$

where $\overline{\Pi}^* \simeq \overline{\Pi}^{**}$, $A \simeq \overline{A}$, $B \simeq \overline{B}$.
(b) in particular:

$$\mathcal{RT}^n((),T,U)(a_1.a_3) = (\overline{\Pi}^1, T^1 \leq Y_{\alpha^1|\beta^1}, \overline{\overline{\Pi}}^3 \rhd_\ell T^4 \leq Y_{\alpha^4|\beta^4}, s)$$

where $\overline{\overline{\Pi}}^3 \simeq \overline{\Pi}^3$, $T^4 \simeq T'$. *Moreover, by the minimality of* a *and by* $a_2 \neq nil$, $\mathcal{RT}^n((),T,U)(a_1.a_3)$ *is not a leaf.*

Proof. As in the previous lemma.□

The next lemma is the first one regarding reduction trees that are not necessarily successful. It states that when two nodes of a generic tree $\mathcal{RT}^n((),T,U)$ compare two similar pairs, then the first one is successful if and only if the second one is successful too.

Lemma 5.26 *For each* $() \rhd_\ell T \leq U \in$ *Start-J and paths* a *and* b *such that*

$$\mathcal{RT}^n((),T,U)(a) = \Pi' \rhd_\ell T' \leq U'$$
$$\mathcal{RT}^n((),T,U)(b) = \Pi'' \rhd_\ell T'' \leq U''$$

where

$$T' \simeq T'' \ \ \textit{and}\ \ U' \simeq U''$$

then $\mathcal{RT}^n((),T,U)(a)$ *is successful (is labeled as success) if and only if the node* $\mathcal{RT}^n((),T,U)(b)$ *is successful too.*

Proof. We only prove that if the node $\mathcal{RT}^n((),T,U)(a)$ is successful then $\mathcal{RT}^n((),T,U)(b)$ is too; the other direction is identical.
Suppose, towards a contradiction, that $\mathcal{RT}^n((),T,U)(a)$ is successful and that $\mathcal{RT}^n((),T,U)(b)$ is not. This means that the second node cannot be proved

by any of the $\Re^{alg\text{-}2}$ rules. Hence the pair of types T'' and U'' are of the form given in the following table:

T''	U''
Top type	type variable
Top type	μ-$\forall$ type
Top type	μ- $\rightarrow$ type
μ-$\forall$ type	μ- $\rightarrow$ type
μ- $\rightarrow$ type	μ-$\forall$ type

In fact when two type variables are compared in a node of a reduction tree, then either the rule ($\text{VarTrans}_{\leq}$) or ($\text{Id}_{\leq}$) is applicable to reduce the prejudgement of the node, hence this is always a successful node. A similar situation holds when one of the two compared types is a recursion variable; in this case either the expansion or the end rules are applicable.

Now, since the shape of a type is preserved by $\simeq$-similarity, we have that if for T'' and U'' one of the cases of the table above holds, then this also holds for T' and U', and this means that $\mathcal{RT}^n((), T, U)(a)$ is a failure node, which is absurd.$\Box$

Observe that Lemma 5.26 does not exclude the case

$$\mathcal{RT}^n((), T, U)(a) = \Pi' \rhd_\ell t_\alpha \leq u_\beta$$
$$\mathcal{RT}^n((), T, U)(b) = \Pi'' \rhd_\ell t_\nu \leq u_\gamma$$

where the first node is successful by ($\text{Id}_{\leq}$) and the second by ($\text{VarTrans}_{\leq}$), or conversely (actually, we will see that this never happens for successful reduction trees). Remember that, for a node, being successful is not the same as being provable: successful means that the node is not a failure node itself (a rule is applicable), while provable means that all the nodes contained in the subtree rooted at that node are successful.

Many of the properties previously stated will be used to prove the following lemma. It states that a successful and expandable tree $\mathcal{RT}^n((), T, U)$ always expands to a successful tree $\mathcal{RT}^{n+1}((), T, U)$. In particular, Corollary 4.17 (used to prove non necessity of renaming) will play a crucial role.

The proof of this lemma is long, but, once it is completed, we have finished our work. Observe that this is the only place where we really exploit the fact that we wait until the third time we meet a pair before stopping, hence this is the lemma that tells us something about the difference between our correct algorithm and the non correct one presented in Section 4.2. The key question we have to address here is: is it possible that two variables t, u are

reduced by the ($\mathrm{Id}_{\leq}$) rule the first time they are met, but, after the expansion of a recursion variable, a similar pair is generated that is reduced by the ($\mathrm{VarTrans}_{\leq}$) rule? This is the problem that may arise when the $\Re^{alg\text{-}1}$ rules are used, and we are going to prove that this never happens in a tree generated by the expansion of a successful $\Re^{alg\text{-}2}$ tree.

Lemma 5.27 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful and expandable tree whose expansion node is*

$$\mathcal{RT}^n((), T, U)(a) = (\overline{\Pi} \rhd_\ell X_{\alpha|\beta} \leq U', s)$$

then $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')$*, where* $T' = \overline{T} \uparrow \beta$ *and* $X_\alpha = \overline{T} \in \mathrm{Left}(\overline{\Pi})$*, is a successful tree. Hence the tree* $\mathcal{RT}^{n+1}((), T, U)$ *is successful too.*

Proof. We have to prove that

$$\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')$$

contains only successful nodes.

From Lemma 5.24, we know that $a = a_1.a_2.a_3$ such that:

$$\begin{aligned}
\mathcal{RT}^n((), T, U)(a_1) &= (\overline{\Pi}^1 \rhd_\ell X_{\alpha^1|\beta^1} \leq U^1, \mathrm{s}) \\
\mathcal{RT}^n((), T, U)(a_1.a_2) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2 \rhd_\ell X_{\alpha^2|\beta^2} \leq U^2, \mathrm{s}) \\
\mathcal{RT}^n((), T, U)(a_1.a_2.a_3) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \\
&\qquad \overline{\Pi}^3 \rhd_\ell X_{\alpha|\beta} \leq U', \mathrm{s}) \\
\mathcal{RT}^n((), T, U)(a_1.a_3) &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^3 \rhd_\ell X_{\alpha^3|\beta^3} \leq U^3, \mathrm{s})
\end{aligned}$$

where:

$$\begin{aligned}
&U^1 \simeq U^2 \simeq U^3 \simeq U' \\
&\overline{\overline{\Pi}}^3 \simeq \overline{\Pi}^3 \\
&\overline{\Pi} = \overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\Pi}^3
\end{aligned}$$

In particular, by Lemma 5.24(2), we know that $\mathcal{RT}^n((), T, U)(a_1.a_3)$ is not a leaf.

Figure 2 illustrates this situation. In the figure, $\Upsilon_{a_1.a_3}$ and $\Upsilon_{a_1.a_2}$ denote the subtrees of $\mathcal{RT}^n((), T, U)$ rooted at the nodes $\mathcal{RT}^n((), T, U)(a_1.a_3)$ and $\mathcal{RT}^n((), T, U)(a_1.a_2)$ respectively. $\Upsilon_{a_1.a_2.a_3}$ denotes the tree $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')$ that extends $\mathcal{RT}^n((), T, U)$ to $\mathcal{RT}^{n+1}((), T, U)$.

Now, as in Lemma 5.24, we will prove that the tree $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')$ (i.e., $\Upsilon_{a_1.a_2.a_3}$) is 'similar' to the subtrees $\Upsilon_{a_1.a_3}$ and $\Upsilon_{a_1.a_2}$. Since both $\Upsilon_{a_1.a_3}$

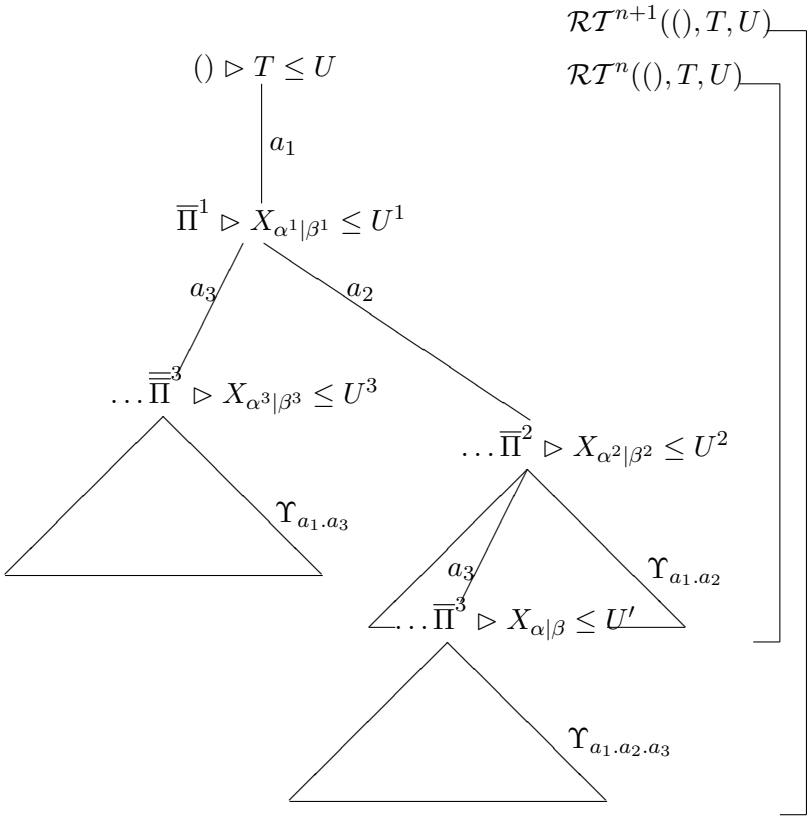


Fig. 2. The subtrees $\Upsilon_{a_1.a_3}$, $\Upsilon_{a_1.a_2}$, $\Upsilon_{a_1.a_2.a_3}$

and $\Upsilon_{a_1.a_2}$ are successful, by Lemma 5.26, we have that the tree $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')$ is successful as well.

Formally, we prove that: $\forall b \in \{0,1,2\}^*$ s.t. $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b) \downarrow$, we have either (1):

$$\begin{aligned}
&\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b) = (\overline{\Pi}, X_{\alpha|\beta} \leq U', \overline{\Pi}^4 \rhd_\ell A \leq B, _\,) \\
&\wedge\ \mathcal{RT}^n((),T,U)(a_1.a_2.0.b) = \\
&\qquad (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\overline{\Pi}}^4 \rhd_\ell \overline{A} \leq \overline{B}, \mathrm{s}) \\
&\wedge\ \mathcal{RT}^n((),T,U)(a_1.a_3.0.b) = \\
&\qquad (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3, \overline{\overline{\Pi}}^4 \rhd_\ell \overline{\overline{A}} \leq \overline{\overline{B}}, \mathrm{s})
\end{aligned}$$

or (2):

$$\begin{aligned}
&\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b) = ((\overline{\Pi}, X_{\alpha|\beta} \leq U')^{-1}, \overline{\Pi}^4 \rhd_\ell A \leq B, _) \\
&\wedge \ \mathcal{RT}^n((), T, U)(a_1.a_2.0.b) = \\
&\qquad ((\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2)^{-1}, \overline{\overline{\Pi}}^4 \rhd_\ell \overline{A} \leq \overline{B}, \mathrm{s}) \\
&\wedge \ \mathcal{RT}^n((), T, U)(a_1.a_3.0.b) = \\
&\qquad ((\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3)^{-1}, \overline{\overline{\Pi}}^4 \rhd_\ell \overline{\overline{A}} \leq \overline{\overline{B}}, \mathrm{s})
\end{aligned}$$

where $A \simeq \overline{A} \simeq \overline{\overline{A}},\ B \simeq \overline{B} \simeq \overline{\overline{B}},\ \overline{\Pi}^3 \simeq \overline{\overline{\Pi}}^3,\ \overline{\Pi}^4 \simeq \overline{\overline{\Pi}}^4 \simeq \overline{\overline{\Pi}}^4$

We prove case (1), by induction on $|b|$ (case (2) is similar).

The case $|b| = 0$ is trivial. If $|b| > 0$ then $b = b'.\bar{b}$ where $\bar{b} \in \{0, 1, 2\}$. By the induction hypothesis we have that the similarity relation holds with respect to the path b'. The proof of the induction step is by cases on the shape of the types compared in $\mathcal{RT}^n((), T, U)(a_1.a_2.0.b')$, $\mathcal{RT}^n((), T, U)(a_1.a_3.0.b')$ and $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b')$. The only interesting case is when two type variables are compared; in the other cases the similarity is proved as in Lemma 5.24. Hence we assume that:

$$\begin{aligned}
\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b') &= (\overline{\Pi}, X_{\alpha|\beta} \leq U', \overline{\Pi}^4 \rhd_\ell t_\nu \leq u_\eta, \mathrm{s}) \\
\mathcal{RT}^n((), T, U)(a_1.a_2.0.b') &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\overline{\Pi}}^4 \\
&\qquad \rhd_\ell t_\gamma \leq u_\delta, \mathrm{s}) \\
\mathcal{RT}^n((), T, U)(a_1.a_3.0.b') &= (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3, \overline{\overline{\Pi}}^4 \\
&\qquad \rhd_\ell t_\tau \leq u_\upsilon, \mathrm{s})
\end{aligned}$$

where

$$\begin{aligned}
&\overline{\Pi} = \overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\Pi}^3 \\
&\overline{\Pi}^3 \simeq \overline{\overline{\Pi}}^3 \\
&\overline{\Pi}^4 \simeq \overline{\overline{\Pi}}^4 \simeq \overline{\overline{\Pi}}^4
\end{aligned}$$

This situation is illustrated by Figure 3.

By hypothesis we have that $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b'.\bar{b}) \downarrow$ and this means that the pre-judgement in b' is reduced to the pre-judgement in $b'.\bar{b}$ by rule (VarTrans$_\leq$). Now the proof continues by distinguishing the following two

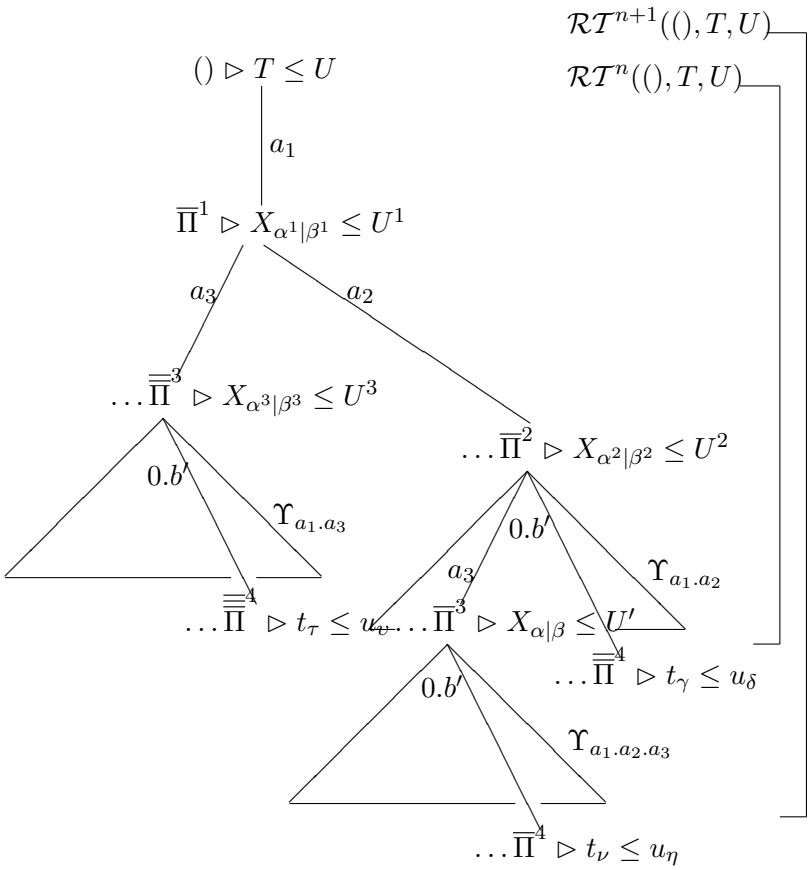


Fig. 3. Type variables comparison.

possible cases:

$$
\begin{aligned}
&(i)\ \ t_\nu \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^3, X_{\alpha|\beta} \leq U', \overline{\Pi}^4))\\
&(ii)\ t_\nu \notin \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^3, X_{\alpha|\beta} \leq U', \overline{\Pi}^4))\\
&\qquad\qquad (\text{that is } t_\nu \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2)))
\end{aligned}
$$

In a nutshell, in case (i) we will show that $t_\tau \leq u_\upsilon$ is also reduced by (VarTrans$_\leq$), and $t_\gamma \leq u_\delta$ will follow by Lemma 5.20, and, similarly, in case (ii) we will show that $t_\gamma \leq u_\delta$ is also reduced by (VarTrans$_\leq$), and $t_\tau \leq u_\upsilon$ will follow by Lemma 5.20.

Suppose we are in case (i). To simplify the notation we let

$$
\begin{aligned}
\overline{\Pi}^{3.4} &= \overline{\Pi}^3, X_{\alpha|\beta} \leq U', \overline{\Pi}^4\\
\overline{\overline{\Pi}}^{3.4} &= \overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3, \overline{\overline{\Pi}}^4
\end{aligned}
$$

which implies $\overline{\Pi}^{3.4} \simeq \overline{\overline{\Pi}}^{3.4}$ (recall that t_ν is defined in $\overline{\Pi}^{3.4}$, hence t_τ is defined in $\overline{\overline{\Pi}}^{3.4}$).

If we suppose, towards a contradiction, that $\mathcal{RT}^n((), T, U)(a_1.a_3.0.b')$ is a leaf proved by $(\mathrm{Id}_{\leq})$, we have

$$\overline{\overline{\Pi}}^{3.4} = \overline{\overline{\Pi}}_1^{3.4}, (t_\tau, u_\upsilon) \leq (A, B), \overline{\overline{\Pi}}_2^{3.4}$$

where, by Corollary 4.17,

$$\nexists \rho \text{ s.t. } t_\rho \in \mathrm{Def}(\mathrm{Left}(\overline{\overline{\Pi}}_2^{3.4}))$$
$$\nexists \rho \text{ s.t. } u_\rho \in \mathrm{Def}(\mathrm{Right}(\overline{\overline{\Pi}}_2^{3.4}))$$

Since $\overline{\Pi}^{3.4} \simeq \overline{\overline{\Pi}}^{3.4}$, in this case we have

$$\overline{\Pi}^{3.4} = \overline{\Pi}_2^{3.4}, (t_{\overline{\nu}}, u_{\overline{\eta}}) \leq (A', B'), \overline{\Pi}_2^{3.4}$$
$$\nexists \rho \text{ s.t. } t_\rho \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}_2^{3.4}))$$
$$\nexists \rho \text{ s.t. } u_\rho \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}_2^{3.4}))$$

and this implies that $\overline{\nu} = \nu$ and $\overline{\eta} = \eta$, therefore the pre-judgement in $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b')$ could be proved by $(\mathrm{Id}_{\leq})$, but this contradicts the hypothesis stating that

$$\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b'.\bar{b}) \downarrow$$

Hence it must be that the node $\mathcal{RT}^n((), T, U)(a_1.a_3.0.b')$ is not a leaf and it is reduced to $\mathcal{RT}^n((), T, U)(a_1.a_3.0.b'.\bar{b})$ by $(\mathrm{VarTrans}_{\leq})$. So we have

$$\overline{\overline{\Pi}}^{3.4} = \overline{\overline{\Pi}}_1^{3.4}, (t_\tau, u^1_{\upsilon^1}) \leq (t^2_{\tau^2}, u^2_{\upsilon^2}), \overline{\overline{\Pi}}_2^{3.4}$$

and

$$\mathcal{RT}^n((), T, U)(a_1.a_3.0.b'.\bar{b}) = (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^{3.4} \rhd_\ell t^2_{\tau^2} \leq u_\upsilon, \mathrm{s})$$

which, by Lemma 5.20, implies that

$$\mathcal{RT}^n((), T, U)(a_1.a_2.0.b'.\bar{b}) =$$
$$(\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2, X_{\alpha^2|\beta^2} \leq U^2, \overline{\overline{\Pi}}^4 \rhd_\ell t^2_{\gamma^2} \leq u_\delta, \mathrm{s})$$

Now, by $\overline{\Pi}^{3.4} \simeq \overline{\overline{\Pi}}^{3.4}$ and by what was stated before, we have that

$$\overline{\Pi}^{3.4} = \overline{\Pi}_1^{3.4}, (t_\nu, u^1_{\eta^1}) \leq (t^2_{\nu^2}, u^2_{\eta^2}), \overline{\Pi}_2^{3.4}$$

which completes the proof of case (i) because this means that

$$\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b'.\bar{b}) = (\overline{\Pi}, X_{\alpha|\beta} \leq U', \overline{\Pi}^4 \rhd_\ell t^2_{\nu^2} \leq u_\eta, \mathrm{s})$$

The remaining case to be considered is (ii):

$$(ii)\ t_\nu \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2))$$

We are going to prove that, in this case, t_γ, t_ν, u_δ, u_η, are all defined in $\overline{\Pi}^{1.2} =_{\mathrm{def}} \overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2$.

Property (ii), by Corollary 4.17, implies that

$$\nexists \rho \text{ s.t. } t_\rho \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^3, X_{\alpha|\beta} \leq U', \overline{\Pi}^4))$$

and, by $\overline{\Pi}^3 \simeq \overline{\overline{\Pi}}^3$ and $\overline{\Pi}^4 \simeq \overline{\overline{\Pi}}^4 \simeq \overline{\overline{\overline{\Pi}}}^4$, that

$$\nexists \rho \text{ s.t. } t_\rho \in \mathrm{Def}(\mathrm{Left}(\overline{\overline{\Pi}}^4))$$
$$\nexists \rho \text{ s.t. } t_\rho \in \mathrm{Def}(\mathrm{Left}(\overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3, \overline{\overline{\overline{\Pi}}}^4))$$

This implies that we have

$$t_\tau \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^1))$$
$$t_\gamma \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^1,\ X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2))$$

hence, by Lemma 5.14, we also have

$$u_\upsilon \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}^1))$$
$$u_\delta \in\ \mathrm{Def}(\mathrm{Right}(\overline{\Pi}^1,\ X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2))$$

In particular, by Corollary 4.17, $u_\upsilon \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}^1))$ implies that

$$\nexists \rho \text{ s.t. } u_\rho \in \mathrm{Def}(\mathrm{Right}(\overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3, \overline{\overline{\overline{\Pi}}}^4))$$

and, by $\overline{\Pi}^3 \simeq \overline{\overline{\Pi}}^3$ and $\overline{\Pi}^4 \simeq \overline{\overline{\overline{\Pi}}}^4$, that:

$$\nexists \rho \text{ s.t. } u_\rho \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}^3, X_{\alpha|\beta} \leq U', \overline{\Pi}^4))$$

hence we have

$$u_\eta \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2))$$

So, if we consider $\overline{\Pi}^{1.2} = \overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\Pi}^2$, we have that

$$t_\gamma,\ t_\nu \in \mathrm{Def}(\mathrm{Left}(\overline{\Pi}^{1.2}))$$
$$u_\delta,\ u_\eta \in \mathrm{Def}(\mathrm{Right}(\overline{\Pi}^{1.2}))$$

and thus, by Lemma 5.21, $t_\gamma = t_\nu$ and $u_\delta = u_\eta$.

Moreover, by hypothesis, we have that $\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b')$ is not a leaf and that it is reduced to

$$\mathcal{RT}((\overline{\Pi}, X_{\alpha|\beta} \leq U'), T', U')(b'.\bar{b}) = (\overline{\Pi}, X_{\alpha|\beta} \leq U', \overline{\Pi}^4 \rhd_\ell t^2_{\nu^2} \leq u_\eta, \mathrm{s})$$

by (VarTrans$_\leq$). Since $t_\gamma = t_\nu$ and $u_\delta = u_\eta$, $\mathcal{RT}^n((), T, U)(a_1.a_2.0.b')$ is reduced to

$$\mathcal{RT}^n((), T, U)(a_1.a_2.0.b'.\bar{b}) = (\overline{\Pi}^{1.2}, X_{\alpha^2|\beta^2} \leq U^2, \overline{\overline{\Pi}}^4 \rhd_\ell t^2_{\nu^2} \leq u_\eta, \mathrm{s}) \quad (*)$$

by (VarTrans$_\leq$). Now, we can apply Lemma 5.20 to the induction hypothesis on

$$\mathcal{RT}^n((), T, U)(a_1.a_2.0.b')$$

and $\mathcal{RT}^n((), T, U)(a_1.a_3.0.b')$ and to (*), so to obtain:

$$\mathcal{RT}^n((), T, U)(a_1.a_3.0.b'.\bar{b}) = \\ (\overline{\Pi}^1, X_{\alpha^1|\beta^1} \leq U^1, \overline{\overline{\Pi}}^3, X_{\alpha^3|\beta^3} \leq U^3, \overline{\overline{\Pi}}^4 \rhd_\ell t^2_{\tau^2} \leq u_\upsilon, \mathrm{s}).$$

This concludes the proof of the last case (ii), hence we have proved the similarity of the subtree $\Upsilon_{a_1.a_2.a_3}$ with respect to the subtrees $\Upsilon_{a_1.a_3}$ and $\Upsilon_{a_1.a_2}$.□

Let us say that two comparisons are "corresponding" if they are generated by the same sequence of rule applications, starting with two similar comparisons $X \leq T$, in the same proof tree. By Lemma 5.26, if a subtree rooted in $X \leq T$ succeeds (up to the next instance of a similar pair) and another subtree, rooted in a similar pair, fails, then a variable-variable pair must be reduced by (Id$_\leq$) in the first subtree, and the corresponding pair in the second subtree must be reduced by (Trans$_\leq$). For example, in the judgement of Section 4.2, the var-var pair $t_\alpha \leq u_\eta$ s reduced by (Id$_\leq$) at step (6), and the corresponding var-var pair $t_\upsilon \leq u_\eta$ is reduced by (Trans$_\leq$) at step (13), and then fails. Intuitively, $t_\upsilon \leq u_\eta$ behaves differently from $t_\alpha \leq u_\eta$ because, in the transition from one comparison to the other, the t variable changed from t_α to t_υ, while u is u_η in both comparisons. In both $t_\alpha \leq u_\eta$ and $t_\upsilon \leq u_\eta$ comparisons, the t variable has been defined three steps before, hence it is a different variable, but with the same De Bruijn index, while u is the same variable, but with a different De Bruijn index. Let us say that, with respect to these pair of comparisons, t is "De Bruijn fixed" while u is "fixed". Our proof shows, essentially, that if we find two successive corresponding var-var comparisons that are both solved by (Id$_\leq$), this is enough to conclude that the compared variables are both "De Bruijn-fixed", or are both "fixed", and that they will be so with respect to all other corresponding pairs, and hence their comparison will always by solved

by ($\mathrm{Id}_{\leq}$). Hence, when we have completely explored two successive similar subtrees, we can safely stop. The end of the second exploration is marked by the third appearance (modulo similarity) of the same compared pair. This explains the magic three.

Lemma 5.28 *If* $() \rhd_\ell T \leq U \in$ *Start-J and* $\mathcal{RT}^n((), T, U)$ *is a successful and expandable tree whose expansion node is*

$$\mathcal{RT}^n((), T, U)(a) = (\overline{\Pi} \rhd_\ell T' \leq Y_{\alpha|\beta}, s)$$

then the tree $\mathcal{RT}((\overline{\Pi}, T' \leq Y_{\alpha|\beta}), T', U')$*, where* $U' = \overline{U} \uparrow \beta$ *and* $Y_\alpha = \overline{U} \in \mathrm{Right}(\overline{\Pi})$*, is a successful tree. Hence the tree* $\mathcal{RT}^{n+1}((), T, U)$ *is successful too.*

Proof. As in the previous lemma.□

Now we can prove soundness.

Theorem 5.29 (Soundness) *For each* $() \rhd_\ell T \leq U \in$ *Start-J*:

$$() \vdash_\ell^{\text{alg-2}} T \leq U \;\Rightarrow\; () \vdash_\ell^{\infty} T \leq U$$

Proof. We have to prove that if $\mathcal{RT}((), T, U)$ is a successful tree then so is $\mathcal{RT}_\infty((), T, U)$. By Lemma 5.8 this amounts to showing that each tree of the sequence (see Definition 5.6)

$$\{\mathcal{RT}^n(\Pi, T, U)\}_{n \in \mathbb{N}}$$

is successful, and this follows by induction on n, by observing that

$$\mathcal{RT}^0(\Pi, T, U) = \mathcal{RT}((), T, U)$$

and by applying Lemmas 5.27 and 5.28 to prove that, for each n, if the tree $\mathcal{RT}^n(\Pi, T, U)$ is successful then $\mathcal{RT}^{n+1}(\Pi, T, U)$ is successful too.□

In the following theorem we prove completeness; unlike soundness, the proof does not present any particular problems.

Theorem 5.30 (Completeness) *For each* $() \rhd_\ell T \leq U \in$ *Start-J:*

$$() \vdash_\ell^{\infty} T \leq U \;\Rightarrow\; () \vdash_\ell^{\text{alg-2}} T \leq U$$

Proof. We have to prove that if $\mathcal{RT}_\infty((), T, U)$ is a successful tree then so is $\mathcal{RT}$ $((), T, U)$. This fact follows by Lemma 5.8 and by observing that $\mathcal{RT}((), T, U) = \mathcal{RT}^0(\Pi, T, U)$.□

6 Transitivity

In this section we prove that the subtype relation we defined on recursive kernel Fun is transitive. We decided not to deal here with the terms of the language kernel Fun and with their reduction relation, since the standard proof of their basic property (subject reduction) works for recursive kernel Fun as well, provided that transitivity holds. Hence, in this section we provide the basic lemma needed to prove that, for recursive kernel Fun, well-typed programs do not go wrong.

We first prove the transitivity of the labeled version of the system, namely that for each $T, U, V \in$ LR-Types such that the three pre-judgements $() \rhd_\ell T \leq U$, $() \rhd_\ell U \leq V$, $() \rhd_\ell T \leq V$ are in Start-J:

$$() \vdash_\ell^\infty T \leq U \text{ and } () \vdash_\ell^\infty U \leq V \;\Rightarrow\; () \vdash_\ell^\infty T \leq V$$

Then, transitivity of recursive kernel Fun will follow from the equivalence of the two systems (Theorem 3.27).

To simplify the notation, in this section we will omit the success and failure labels in the nodes of reduction trees. We start with a couple of lemmas regarding type variables.

Lemma 6.1 *For each* $() \rhd_\ell T \leq U \in$ *Start-J, for any* a *such that*

$$\mathcal{RT}^\infty((), T, U)(a) = (\Pi \rhd_\ell t_\alpha \leq u_\beta)$$

the subtree rooted at $\mathcal{RT}^\infty((), T, U)(a)$ *is successful* $\Leftrightarrow$

$$\exists\, t'_\eta \in \text{Bounds}(t_\alpha,\ \text{Left}(\Pi)) \;\; s.t.\ (t'_\eta, u_\beta) \in \text{Def}(\Pi)$$

Proof.
($\Rightarrow$) By induction on the depth of the subtree rooted at $\mathcal{RT}^\infty((), T, U)(a)$.
($\Leftarrow$) By induction on #Bounds(t_α, Left(Π)).□

Definition 6.2 *For each* $() \rhd_\ell T \leq U$ *in Start-J, a path* $\overline{a}$ *such that*

$$\mathcal{RT}^\infty((), T, U)(\overline{a}) \downarrow$$

is called var-var-free *if*

$$\nexists\, a' \prec_p \overline{a} \text{ such that } \mathcal{RT}^\infty((), T, U)(a') = (\Pi \rhd_\ell t_\alpha \leq u_\beta)$$

Note that $\mathcal{RT}^\infty((), T, U)(\overline{a}) = (\Pi \rhd_\ell t_\alpha \leq u_\beta)$ *is not excluded by the definition.*

Lemma 6.3 *For each* $() \rhd_\ell T \leq U$, $() \rhd_\ell U \leq V$, $() \rhd_\ell T \leq V$ *in Start-J such that* $() \vdash^\infty_\ell T \leq U$ *and* $() \vdash^\infty_\ell U \leq V$, *and for each var-var-free path* a *such that*

$$\mathcal{RT}^\infty((), T, V)(a) = (\Pi' \rhd_\ell T' \leq V')$$

there exist two paths a_1 *and* a_2 *and a type* U' *such that*

$$\begin{aligned}\mathcal{RT}^\infty((), T, U)(a_1) &= (\Pi'' \rhd_\ell T' \leq U')\\ \mathcal{RT}^\infty((), U, V)(a_2) &= (\Pi''' \rhd_\ell U' \leq V')\end{aligned}$$

Moreover the three bi-environments Π', Π'', Π''' *satisfy the following properties*

1. $\forall\, (t_\alpha, u_\beta) \in \mathrm{Def}(\Pi'').\forall\, (u_\beta, v_\delta) \in \mathrm{Def}(\Pi''').$

   $$\begin{aligned}&(t_\alpha, v_\delta) \in \mathrm{Def}(\Pi')\\ &\mathrm{Bounds}(t_\alpha,\ \mathrm{Left}(\Pi'')) = \mathrm{Bounds}(t_\alpha,\ \mathrm{Left}(\Pi'))\\ &\mathrm{Bounds}(u_\beta,\ \mathrm{Right}(\Pi'')) = \mathrm{Bounds}(u_\beta,\ \mathrm{Left}(\Pi'''))\\ &\mathrm{Bounds}(v_\delta,\ \mathrm{Right}(\Pi''')) = \mathrm{Bounds}(v_\delta,\ \mathrm{Right}(\Pi'))\end{aligned}$$

2. $X_\alpha = A \in \mathrm{Right}(\Pi'') \Leftrightarrow X_\alpha = A \in \mathrm{Left}(\Pi''')$
3. $X_\alpha = A \in \mathrm{Left}(\Pi') \Leftrightarrow X_\alpha = A \in \mathrm{Left}(\Pi'')$
4. $X_\alpha = A \in \mathrm{Right}(\Pi') \Leftrightarrow X_\alpha = A \in \mathrm{Right}(\Pi''')$

Proof. By induction on $|a|$. The case $a = nil$ is trivial. Suppose that $a = a'.\overline{a}$, with $\overline{a} \in \{0,\ 1,\ 2\}$, and assume that the lemma holds for the path a'. The most difficult case is $\overline{a} = 2$, the other ones are similar or even simpler. When $\overline{a} = 2$ the node $\mathcal{RT}^\infty((), T, V)(a)$ has been reduced from $\mathcal{RT}^\infty((), T, V)(a')$ by $(\forall_\leq)$ (see Definition 5.2). Formally:

$$\begin{aligned}\mathcal{RT}^\infty((), T, V)(a') &= (\overline{\Pi}' \rhd_\ell (\mu X_\alpha.\forall t_\alpha \leq A.T') \leq (\mu Y_\beta.\forall v_\beta \leq B.V'))\\ \mathcal{RT}^\infty((), T, V)(a'.\overline{a}) &= (\Pi' \rhd_\ell T' \leq V')\end{aligned}$$

where

$$\Pi' = \overline{\Pi}',\ (X_\alpha = \mu X_\alpha.\forall t_\alpha \leq A.T',\ Y_\beta = \mu Y_\beta.\forall v_\beta \leq B.V'),\ (t_\alpha, v_\beta) \leq (A,B)$$

By the induction hypothesis we have that there exist two paths a'_1 and a'_2 and a type U'' such that

$$\begin{aligned}\mathcal{RT}^\infty((), T, U)(a'_1) &= (\overline{\Pi}'' \rhd_\ell (\mu X_\alpha.\forall t_\alpha \leq A.T') \leq U'')\\ \mathcal{RT}^\infty((), U, V)(a'_2) &= (\overline{\Pi}''' \rhd_\ell U'' \leq (\mu Y_\beta.\forall v_\beta \leq B.V'))\end{aligned}$$

and that the three bi-environments $\overline{\Pi}'$, $\overline{\Pi}''$, $\overline{\Pi}'''$ satisfy properties 1, 2, 3, 4. Since $\mathcal{RT}^\infty((), T, U)$ and $\mathcal{RT}^\infty((), U, V)$ are successful, the type U'' is either a μ-$\forall$ type or a recursion variable that unfolds to a μ-$\forall$ type. We consider only the second case because it is slightly more difficult than the first one.
Hence, suppose that $U'' = K_{\tau|\theta}$. We have that

$$\begin{array}{l}\mathcal{RT}^\infty((), T, U)(a'_1.0) = \\ \quad (\overline{\Pi}'',\ (\mu X_\alpha.\forall t_\alpha \leq A.T' \leq K_{\tau|\theta}) \rhd_\ell\ \mu X_\alpha.\forall t_\alpha \leq A.T'\ \leq \mu K_\theta.\forall s_\theta \leq C.D) \\ \mathcal{RT}^\infty((), U, V)(a'_2.0) = \\ \quad (\overline{\Pi}''',\ (K_{\tau|\theta} \leq \mu Y_\beta.\forall v_\beta \leq B.V') \rhd_\ell\ \mu K_\theta.\forall s_\theta \leq C.D\ \leq \mu Y_\beta.\forall v_\beta \leq B.V')\end{array}$$

where $\mu K_\theta.\forall s_\theta \leq C.D = (\mu K_\tau.\forall s_\tau \leq C'.D') \uparrow \theta$ with

$$(K_\tau = \mu K_\tau.\forall s_\tau \leq C'.D') \in \text{ Right}(\overline{\Pi}'').$$

Moreover, we have that

$$\begin{array}{lcl}\mathcal{RT}^\infty((), T, U)(a'_1.0.2) & = & (\overline{\overline{\Pi}}'' \rhd_\ell T' \leq D) \\ \mathcal{RT}^\infty((), U, V)(a'_2.0.2) & = & (\overline{\overline{\Pi}}''' \rhd_\ell D \leq V')\end{array}$$

with

$$\begin{array}{ll}\Pi'' = & \overline{\Pi}'',\ (\mu X_\alpha.\forall t_\alpha \leq A.T' \leq K_{\tau|\theta}), \\ & (X_\alpha = \mu X_\alpha.\forall t_\alpha \leq A.T',\ K_\theta = \mu K_\theta.\forall s_\theta \leq C.D),\ (t_\alpha, s_\theta) \leq (A, C) \\ \Pi''' = & \overline{\Pi}''',\ (K_{\tau|\theta} \leq \mu Y_\beta.\forall v_\beta \leq B.V'), \\ & (K_\theta = \mu K_\theta.\forall s_\theta \leq C.D,\ Y_\beta = \mu Y_\beta.\forall v_\beta \leq B.V'),\ (s_\theta, v_\beta) \leq (C, B)\end{array}$$

Now, by the induction hypothesis and by the definition of Π', Π'' and Π'', it is easy to prove that these bi-environments satisfy properties 1, 2, 3, 4.□

Theorem 6.4 *For each* $() \rhd_\ell T \leq U$, $() \rhd_\ell U \leq V$, $() \rhd_\ell T \leq V$ *pre-judgements in Start-J,*

$$() \vdash^\infty_\ell T \leq U \text{ and } () \vdash^\infty_\ell U \leq V\ \Rightarrow\ () \vdash^\infty_\ell T \leq V$$

Proof. We split the proof into two parts. In the first one we prove that all the nodes in $\mathcal{RT}^\infty((), T, V)$ that correspond to a var-var-free path a are successful nodes of $\mathcal{RT}^\infty((), T, V)$. Then, in the second part, we prove that all the subtrees of a reduction tree

$$\mathcal{RT}^\infty((), T, V)(a) = (\Pi' \rhd_\ell t_\alpha \leq v_\beta)$$

are successful.

(1) Suppose, towards a contradiction, that for a var-var-free path a we have a failure node (which is actually a leaf) of $\mathcal{RT}^{\infty}((), T, V)$. Formally we have that

$$\mathcal{RT}^{\infty}((), T, V)(a) = (\Pi' \rhd_{\ell} T' \leq V')$$

where the shape of T' and V' corresponds to one of the cases of the following table

T'	V'
Top type	type variable
Top type	μ-$\forall$ type
Top type	μ- $\rightarrow$ type
μ-$\forall$ type	μ- $\rightarrow$ type
μ- $\rightarrow$ type	μ-$\forall$ type

The first three cases can be synthesized in

$$T' = \top \text{ and } V' \neq \top$$

In this case, by Lemma 6.3, we have that

$$\begin{aligned} \mathcal{RT}^{\infty}((), T, U)(a_1) &= (\Pi'' \rhd_{\ell} \top \leq U') \\ \mathcal{RT}^{\infty}((), U, V)(a_2) &= (\Pi''' \rhd_{\ell} U' \leq V') \end{aligned}$$

Since $\mathcal{RT}^{\infty}((), T, U)$ is successful, it must be that $U' = \top$, but this implies that $\mathcal{RT}^{\infty}((), U, V)(a_2)$ is a failure node because of $V' \neq \top$ and this would contradict the fact that $\mathcal{RT}^{\infty}((), U, V)$ is successful.
Now suppose that

$$T' = \mu X_{\alpha}.\forall t_{\alpha} \leq T''.T''' \text{ and } V' = \mu Y_{\beta}.V'' \rightarrow V'''$$

By Lemma 6.3 we have that

$$\begin{aligned} \mathcal{RT}^{\infty}((), T, U)(a_1) &= (\Pi'' \rhd_{\ell} \mu X_{\alpha}.\forall t_{\alpha} \leq T''.T''' \leq U') \\ \mathcal{RT}^{\infty}((), U, V)(a_2) &= (\Pi''' \rhd_{\ell} U' \leq \mu Y_{\beta}.V'' \rightarrow V''') \end{aligned}$$

and, since $\mathcal{RT}^{\infty}((), T, U)(a_1)$ is successful, it must be that

$$(1)\ U' = \mu Z_{\eta}.\forall s_{\eta} \leq U''.U''' \ \textit{or}\ (2)\ U' = \top \ \textit{or}\ (3)\ U' = Z_{\nu|\eta}$$

and $Z_{\nu|\eta}$ expands to a type $\mu Z_{\eta}.\forall s_{\eta} \leq U''.U'''$ after an application of the (RUnf$_{\leq}$) rule.

In the first two cases we immediately have that $\mathcal{RT}^\infty((), U, V)(a_2)$ is a failure node, while in the third case we have that $\mathcal{RT}^\infty((), U, V)(a_2)$ is reduced to a failure node after an application of the $(\mathrm{LUnf}_\leq)$ rule, since, by Lemma 6.3(2), for this node too the variable $Z_{\nu|\eta}$ expands to a type $\mu Z_\eta.\forall s_\eta \leq U''.U'''$; hence in both cases we reach a contradiction.
The case

$$T' = \mu Y_\alpha.T'' \to T''' \quad \text{and} \quad V' = \mu X_\beta.\forall t_\beta \leq V''.V'''$$

is analogous to the one just proved.

(2) Now we prove that all the subtrees of a node

$$\mathcal{RT}^\infty((), T, V)(a) = (\Pi' \rhd_\ell t_\alpha \leq v_\beta)$$

where a is a var-var-free path, are successful.
Suppose that for a var-var-free path a we have that

$$\mathcal{RT}^\infty((), T, V)(a) = (\Pi' \rhd_\ell t_\alpha \leq v_\beta)$$

By Lemma 6.3 we have that

$$\begin{aligned} \mathcal{RT}^\infty((), T, U)(a_1) &= (\Pi'' \rhd_\ell t_\alpha \leq u_\eta) \\ \mathcal{RT}^\infty((), U, V)(a_2) &= (\Pi''' \rhd_\ell u_\eta \leq v_\beta) \end{aligned}$$

and by assumption and Lemma 6.1:

$$\begin{aligned} &\exists\, t'_\delta. \quad t'_\delta \in \mathrm{Bounds}(t_\alpha,\ \mathrm{Left}(\Pi'')) \ (\mathrm{a}') \\ &\qquad \text{and} \ \ (t'_\delta, u_\eta) \in \mathrm{Def}(\Pi'') \ (\mathrm{a}'') \\ &\exists\, u'_\lambda. \quad u'_\lambda \in \mathrm{Bounds}(u_\eta,\ \mathrm{Left}(\Pi''')) \ (\mathrm{b}') \\ &\qquad \text{and} \ \ (u'_\lambda, v_\beta) \in \mathrm{Def}(\Pi''') \ (\mathrm{b}'') \end{aligned}$$

Now we observe the following facts

(c) $\text{Bounds}(u'_\lambda,\ \text{Left}(\Pi'''))\subseteq\text{Bounds}(u_\eta,\ \text{Left}(\Pi'''))$
by (b′)

(d) $\text{Bounds}(u_\eta,\ \text{Left}(\Pi'''))=\text{Bounds}(u_\eta,\ \text{Right}(\Pi''))$
by Lemma 6.3

(f) $u'_\lambda\in\text{Bounds}(u_\eta,\ \text{Right}(\Pi''))$
by (b′) and (d)

(g) $\exists\, t''_\theta\in\text{Bounds}(t'_\delta,\ \text{Left}(\Pi''))$ (g1) s.t.
$(t''_\theta,u'_\lambda)\in\text{Def}(\Pi'')$ (g2)
by (a″), (f) and Lemma 5.17

(h) $t''_\theta\in\text{Bounds}(t_\alpha,\ \text{Left}(\Pi''))$
by (g1) and (a′)

(i) $\text{Bounds}(t_\alpha,\ \text{Left}(\Pi''))=\text{Bounds}(t_\alpha,\ \text{Left}(\Pi'))$
by Lemma 6.3(1)

(l) $t''_\theta\in\text{Bounds}(t_\alpha,\ \text{Left}(\Pi'))$
by (h) and (i)

(m) $(t''_\theta,v_\beta)\in\text{Def}(\Pi')$
by (g2), (b″) and Lemma 6.3(1)

By (l), (m) and Lemma 6.1 we can conclude that the subtree rooted at $\mathcal{RT}^\infty((),T,V)(a)$ is successful.□

Theorem 6.5 (Transitivity) *For each closed pre-judgements* $()\rhd T\le U$, $()\rhd U\le V$ *and* $()\rhd T\le V$ *in recursive kernel Fun,*

$$()\vdash T\le U\ \wedge\ ()\vdash U\le V\ \ \Rightarrow\ \ ()\vdash T\le V$$

Proof.

By hypothesis: $() \vdash T \leq U \ \wedge \ () \vdash U \leq V$

By Corollary 3.29,

for any α: $() \vdash_{\ell}^{\infty} [\{\}, \alpha](T) \leq [\{\}, \alpha](U) \wedge$
$() \vdash_{\ell}^{\infty} [\{\}, \alpha](U) \leq [\{\}, \alpha](V)$

By transitivity: $() \vdash_{\ell}^{\infty} [\{\}, \alpha](T) \leq [\{\}, \alpha](V)$

By Corollary 3.28: $() \vdash \text{Erase}([\{\}, \alpha](T)) \leq \text{Erase}([\{\}, \alpha](V))$

i.e. $() \vdash T \leq V$

□

7 Conclusions

We have studied the subtype relation for recursive types in kernel Fun. This problem is important because the combination of subtyping, parametric polymorphism, and recursion is essential in the context of strongly typed object-oriented languages.

The main result of this work has been the definition of a subtyping algorithm for strongly recursive kernel Fun, which was the only one known for this class of languages (building on a preliminary version of this paper [CG99], Jeffrey defined in [Jef01] a new algorithm with quite different features).

We have proved our algorithm to be sound and complete and the proof is technically challenging. We have also been able to prove, by showing non trivial counterexamples, that the most natural algorithm to attack the problem is not complete, and that a natural obvious relaxation is not correct. These two examples explain why we needed to look for a more complex approach. We consider these counterexamples to be important contributions of this work.

We have proved that our algorithm can be defined without variable renaming. This makes the algorithm very efficient in practice since, if variable renaming can be avoided, memory allocation is greatly reduced during the execution of the subtyping algorithm, and the key step of similarity checking can be reduced to pointer equality checking. Moreover, this property may allow the efficient first-order subtype checking algorithm presented in ([KPS93]) to be imported in this context. That algorithm is based on reusing, in a branch of a proof, a comparison $T \leq U$ that has been proved in a different branch. This

technique seems to be very hard to adopt if variables can be renamed. Similarly, [Ghe96] describes a technique that transforms the standard exponential subtype checking algorithm for kernel Fun into a PTime algorithm, crucially exploiting the memorization of already proved judgements. This technique only works in the absence of renaming.

Finally, we have proved that the subtype relation defined by our algorithm is transitive. This is the core of the proof of the subject reduction property for the underlying term language, although, due to lack of space, this problem is not discussed in this paper.

As a consequence of this work, we now feel the need to study the more general field of "regular" trees with variables, i.e. those trees that abstractly describe the terms of a language that combines a μ-like operator with variable binders. Both our counterexamples involve infinite trees-with-variables that can be finitely described but that are not regular in the usual sense of the word. The study of relations coinductively defined on such trees constitutes a generalization of this work which would probably shed light on some of the 'surprising' results we have described here.

Acknowledgements

We would like to thank Paolo Baldan and the anonymous referees for patient reading, precise corrections, and useful suggestions.

References

[AC93] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993.

[AC96] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[ACO85] Antonio Albano, Luca Cardelli, and Renzo Orsini. Galileo: A strongly typed, interactive conceptual language. *ACM Transactions on Database Systems*, 10(2):230–260, 1985.

[AF96] Martín Abadi and Marcelo P. Fiore. Syntactic considerations on recursive types. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 242–252, 1996.

[AGO95] Antonio Albano, Giorgio Ghelli, and Renzo Orsini. Fibonacci: A programming language for object databases. *VLDB Journal*, 4(3):403–444, 1995.

[AN80] A. Arnold and M. Nivat. Metric interpretations of infinite trees

and semantics of non deterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980.

[BCP96] Kim B. Bruce, Luca Cardelli, and Benjamin C. Pierce. Comparing object encodings. In *Proceedings of Theoretical Aspects of Computer Software*, pages 415–438, September 1996.

[BH97] Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. In *Proceedings of Typed Lambda Calculi And Application*, LNCS, pages 63–81, 1997.

[BKR98] N. Benton, A. Kennedy, and G. Russel. Compiling standard ML to java bytecodes. In *Proceedings of ACM SIGPLAN International Conference on Functional Programming*, pages 129–140, 1998.

[Car89] Felice Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of International Colloquium on Automata, Languages, and Programming*, LNCS, pages 164–178, 1989.

[Car90] Luca Cardelli. Notes about $F\omega_{<:}$. Unpublished manuscript, October 1990.

[CG92] P.-L. Curien and Giorgio Ghelli. Coherence of subsumption, minimum typing and type checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2(1):55–91, 1992.

[CG99] Dario Colazzo and Giorgio Ghelli. Subtyping recursive types in kernel Fun. In *Proceedings of IEEE Synposium On Logic In Computer Science*, pages 137–146, 1999.

[CL91] Luca Cardelli and Giuseppe Longo. A semantic basis for Quest. *Journal of Functional Programming*, 1(4):417–458, 1991.

[CMMS94] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. *Information and Computation*, 109(1–2):4–56, 1994. Preliminary version in TACS '91 (Sendai, Japan, pp. 750–770).

[Cop] James O. Coplien. *Pattern Languages of Program Design*. Addison-Wesley.

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[DT88] Scott Danforth and Chris Tomlinson. Type theories and object-oriented programming. *ACM Computing Surveys*, 20(1):29–72, 1988.

[FM96] Kathleen Fisher and John Mitchell. The development of type systems for object-oriented languages. *Theory and Practice of Object Systems*, 1(3):189–220, 1996.

[Ghe90] Giorgio Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism*. PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1990. Tech. Rep. TD-6/90.

[Ghe91] Giorgio Ghelli. Modelling features of object-oriented languages

in second order functional languages with subtypes. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of Foundations of Object-Oriented Languages*, LNCS, pages 311–340, 1991.

[Ghe93] Giorgio Ghelli. Recursive types are not conservative over $F_{\leq}$. In *Proceedings of Typed Lambda Calculi And Application*, LNCS, pages 146–162, 1993.

[Ghe95] Giorgio Ghelli. Divergence of $F_{\leq}$ type checking. *Theoretical Computer Science*, 139(1-2):131–162, 1995.

[Ghe96] Giorgio Ghelli. Complexity of kernel Fun subtype checking. In *Proceedings of ACM SIGPLAN International Conference on Functional Programming*, pages 134–145, 1996.

[GLP00] Vladimir Gapejev, Michael Y. Levin, and Benjamin C. Pierce. Recursive subtyping revealed: functional pearl. In *Proceedings of ACM SIGPLAN International Conference on Functional Programming*, pages 221–231, 2000.

[GM94] Carl A. Gunter and John C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*. The MIT Press, 1994.

[GMW79] M.H. Gordon, R.M. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *LNCS*. Springer Verlag, 1979.

[Gun92] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. The MIT Press, 1992.

[Jef01] A.S.A. Jeffrey. A symbolic labelled transition system for coinductive subtyping of f-mu-sub types. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 323–333, 2001.

[Ken98] Andrew Kennedy, 1998. personal communication.

[KPS93] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient recursive subtyping. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 419–428, 1993.

[KS92] D. Katiyar and S. Sankar. Completely bounded quantification is decidable. In *Proceedings of ACM SIGPLAN Workshop on ML and its Applications*, 1992.

[Nel91] Greg Nelson. *Systems Programming in Modula-3*. Prentice Hall, 1991.

[Pie94] B. C. Pierce. Bounded quantification is undecidable. *Information and Computation*, 112(1):131–165, 1994.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[TU96] Jerzy Tiuryn and Paweł Urzyczyn. The subtyping problem for second-order types is undecidable. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 74–85, 1996.

Appendix A: Recursive Kernel Fun

Syntax

Types	T, U	$::= \top \mid t \mid X \mid \mu X.\forall t \leq T.U \mid \mu X.T \to U$
Pre-Judgements	P	$::= \Gamma \rhd \text{Env} \mid \Gamma \rhd T\ \text{Type} \mid \Pi \rhd T \leq U$
Judgements	J	$::= \Gamma \vdash \text{Env} \mid \Gamma \vdash T\ \text{Type} \mid \Pi \vdash T \leq U$
Bi-Environments	Π	$::= () \mid \Pi, (t, u) \leq (T, U) \mid \Pi, (X = T, Y = U)$
Environments	Γ	$::= () \mid \Gamma, t \leq T \mid \Gamma, X = T$

Good formation rules

$$\frac{}{() \vdash \text{Env}}\ (\Gamma\ \text{EmptyForm}) \qquad \frac{t \notin \text{Def}(\Gamma) \quad \Gamma \vdash T\ \text{Type}}{\Gamma, t \leq T \vdash \text{Env}}\ (\Gamma\ \text{BoundForm})$$

$$\frac{X \notin \text{Def}(\Gamma) \quad \Gamma \vdash T\ \text{Type}}{\Gamma, X = T \vdash \text{Env}}\ (\Gamma\ \text{EqForm})$$

$$\frac{t \leq T \in \Gamma \quad \Gamma \vdash T\ \text{Type}}{\Gamma \vdash t\ \text{Type}}\ (\text{T-VarForm})$$

$$\frac{\Gamma \vdash \text{Env}}{\Gamma \vdash \top\ \text{Type}}\ (\top\ \text{Form}) \qquad \frac{X \in \text{Def}(\Gamma) \quad \Gamma \vdash \text{Env}}{\Gamma \vdash X\ \text{Type}}\ (\text{R-VarForm})$$

$$\frac{\Gamma, X = \top \vdash T\ \text{Type} \quad \Gamma, X = \top \vdash U\ \text{Type}}{\Gamma \vdash \mu X.T \to U\ \text{Type}}\ (\to\ \text{Form})$$

$$\frac{\Gamma, X = \top, t \leq T \vdash U\ \text{Type}}{\Gamma \vdash \mu X.\forall t \leq T.U\ \text{Type}}\ (\forall\ \text{Form})$$

Subtyping rules

$$\frac{\Pi \vdash T, \top\ \text{Type}}{\Pi \vdash T \leq \top}\ (\top_{\leq}) \qquad \frac{(t, u) \in \text{Def}(\Pi) \quad \Pi \vdash t, u\ \text{Type}}{\Pi \vdash t \leq u}\ (\text{Id}_{\leq})$$

$$\frac{\begin{array}{l}(t, u) \leq (T', U') \in \Pi \\ \textit{for all } X.\ (U \neq X) \quad U \neq \top \quad U \neq u \quad \Pi \vdash T' \leq U\end{array}}{\Pi \vdash t \leq U}\ (\text{VarTrans}_{\leq})$$

$$\frac{\begin{array}{l}\Pi' = (\Pi, (X{=}\mu X.T \to U,\ Y{=}\mu Y.T' \to U')) \\ \mathrm{Swap}(\Pi') \vdash T' \leq T \quad \Pi' \vdash U \leq U'\end{array}}{\Pi \vdash \mu X.T \to U \leq \mu Y.T' \to U'} \ (\to_{\leq})$$

$$\frac{\begin{array}{l}\Pi' = (\Pi, (X{=}\mu X.\forall t{\leq}T.U,\ Y{=}\mu Y.\forall t'{\leq}T'.U')) \\ \Pi' \vdash T \triangleq T' \quad \Pi', (t,t'){\leq}(T,T') \vdash U \leq U'\end{array}}{\Pi \vdash \mu X.\forall t{\leq}T.U \leq \mu Y.\forall t'{\leq}T'.U'} \ (\forall_{\leq})$$

$$\frac{X{=}T \in \mathrm{Left}(\Pi) \quad \Pi \vdash\ T{\uparrow}{\leq}\ U}{\Pi \vdash X\ \leq\ U} \ (\mathrm{LUnf}_{\leq})$$

$$\frac{\textit{for all } X.\ (T \neq X) \quad Y{=}U \in \mathrm{Right}(\Pi) \quad \Pi \vdash T\ \leq\ U{\uparrow}}{\Pi \vdash T\ \leq\ Y} \ (\mathrm{RUnf}_{\leq})$$

The symbol $\uparrow T$ in ($\_\mathrm{Unf}_{\leq}$) rules denotes a renaming of both type and recursive defined variables in T.

Appendix B: Labeled Recursive Kernel Fun

Syntax

$$
\begin{array}{lll}
\textbf{Types} & T, U ::= \top_\beta \mid t_{\alpha|\beta} \mid X_{\alpha|\beta} \mid \mu X_\alpha.\forall t_\alpha \leq T.U \\
 & \qquad \mid \mu X_\alpha.T \to U \\
\textbf{Pre-Judgements} & P ::= \Gamma \rhd_\ell \text{Env} \mid \Gamma \rhd_\ell T \text{ Type} \mid \Pi \rhd_\ell T \leq U \\
\textbf{Judgements} & J ::= \Gamma \vdash_\ell \text{Env} \mid \Gamma \vdash_\ell T \text{ Type} \mid \Pi \vdash_\ell T \leq U \\
\textbf{Bi-Environments} & \Pi ::= () \mid \Pi, (t_\alpha, u_\beta) \leq (T, U) \\
 & \qquad \mid \Pi, (X_\alpha = T_{X,\alpha}, Y_\delta = U_{Y,\delta}) \\
 & \qquad \mid \Pi, X_{\alpha|\beta} \diamond T \mid \Pi, T \diamond X_{\alpha|\beta} \\
\textbf{Environments} & \Gamma ::= () \mid \Gamma, t_\alpha \leq T \mid \Gamma, X_\alpha = T_{X,\alpha} \mid \Gamma, T
\end{array}
$$

Good formation rules

$$\frac{}{() \vdash_\ell \text{Env}} \ (\Gamma \text{ EmptyForm}) \qquad \frac{t_\alpha \notin \text{Def}(\Gamma) \quad \Gamma \vdash_\ell T \text{ Type}}{\Gamma, t_\alpha \leq T \vdash_\ell \text{Env}} \ (\Gamma \text{ BoundForm})$$

$$\frac{\Gamma \vdash_\ell T \text{ Type}}{\Gamma, T \vdash_\ell \text{Env}} \ (\Gamma \text{ TypeForm})$$

$$\frac{X_\alpha \notin \text{Def}(\Gamma) \quad \Gamma \vdash_\ell T \text{ Type}}{\Gamma, X_\alpha = T \vdash_\ell \text{Env}} \ (\Gamma \text{ EqForm})$$

$$\frac{t_\alpha \leq T \in \Gamma \quad \Gamma \vdash_\ell T \text{ Type}}{\Gamma \vdash_\ell t_{\alpha|\beta} \text{ Type}} \ (\text{T-VarForm})$$

$$\frac{\Gamma \vdash_\ell \text{Env}}{\Gamma \vdash_\ell \top_\beta \text{ Type}} \ (\top \text{ Form}) \qquad \frac{X_\alpha \in \text{Def}(\Gamma) \quad \Gamma \vdash_\ell \text{Env}}{\Gamma \vdash_\ell X_{\alpha|\beta} \text{ Type}} \ (\text{R-VarForm})$$

$$\frac{\Gamma, X_\alpha = \top_\alpha \vdash_\ell T \text{ Type} \quad \Gamma, X_\alpha = \top_\alpha \vdash_\ell U \text{ Type}}{\Gamma \vdash_\ell \mu X_\alpha.T \to U \text{ Type}} \ (\to \text{ Form})$$

$$\frac{\Gamma, X_\alpha = \top_\alpha, t_\alpha \leq T \vdash_\ell U \text{ Type}}{\Gamma \vdash_\ell \mu X_\alpha.\forall t_\alpha \leq T.U \text{ Type}} \ (\forall \text{ Form})$$

Subtyping rules

$$\frac{\Pi \vdash_\ell T, \top \text{ Type}}{\Pi \vdash_\ell T \leq \top_\beta} \ (\top_\leq) \qquad \frac{(t_\alpha, u_\nu) \in \text{Def}(\Pi) \quad \Pi \vdash_\ell t_{\alpha|\beta}, u_{\nu|\eta} \text{ Type}}{\Pi \vdash_\ell t_{\alpha|\beta} \leq u_{\nu|\eta}} \ (\text{Id}_\leq)$$

$$\frac{(t_\alpha, u_\nu) \leq (T', U') \in \Pi \quad \textit{for all } X_{\theta|\delta}.\ (U \neq X_{\theta|\delta}) \quad U \neq u_{\nu|-} \quad U \neq \top_- \quad \Pi \vdash_\ell T' \leq U}{\Pi \vdash_\ell t_{\alpha|\beta} \leq U} \quad (\mathrm{VarTrans}_\leq)$$

$$\frac{\Pi' = \Pi, (X_\alpha{=}\mu X_\alpha.T \to U,\ \ Y_\nu{=}\mu Y_\nu.T' \to U') \quad \mathrm{Swap}(\Pi') \vdash_\ell T' \ \leq\ T \quad \Pi' \vdash_\ell U \ \leq\ U'}{\Pi \vdash_\ell \mu X_\alpha.T \to U \ \leq\ \mu Y_\nu.T' \to U'} \quad (\to_\leq)$$

$$\frac{\Pi' = \Pi, (X_\alpha{=}\mu X_\alpha.\forall t_\alpha \leq T.U,\ \ Y_\nu{=}\mu Y_\nu.\forall u_\nu \leq T'.U') \quad \Pi' \vdash_\ell T \mathrel{\hat{=}} T' \quad \Pi', (t_\alpha, u_\nu) \leq (T, T') \vdash_\ell U \leq U'}{\Pi \vdash_\ell \mu X_\alpha.\forall t_\alpha \leq T.U \ \leq\ \mu Y_\nu.\forall u_\nu \leq T'.U'} \quad (\forall_\leq)$$

$$\frac{X_\alpha{=}T \in \mathrm{Left}(\Pi) \quad \Pi, X_{\alpha|\beta} \leq U \vdash_\ell T \uparrow \beta \ \leq\ U}{\Pi \vdash_\ell X_{\alpha|\beta} \ \leq\ U} \quad (\mathrm{LUnf}_\leq)$$

$$\frac{\textit{for all } X_{\theta|\delta}.\ (T \neq X_{\theta|\delta}) \quad Y_\nu{=}U \in \mathrm{Right}(\Pi) \quad \Pi, T \leq Y_{\nu|\eta} \vdash_\ell T \ \leq\ U \uparrow \eta}{\Pi \vdash_\ell T \ \leq\ Y_{\nu|\eta}} \quad (\mathrm{RUnf}_\leq)$$

The rules of our sound and complete algorithm are obtained by adding two new termination rules to the $\Re^\infty$ system and by modifying the unfolding rules as follows.

$$\frac{X_{\alpha|\beta} \leq U \ \in_2^\simeq \Pi \quad \Pi \vdash_\ell X_{\alpha|\beta}, U\ \mathrm{Type}}{\Pi \vdash_\ell X_{\alpha|\beta} \ \leq\ U} \quad (\mathrm{LEnd}_\leq)$$

$$\frac{T \ \leq\ Y_{\nu|\eta} \ \in_2^\simeq \Pi \quad \Pi \vdash_\ell T, Y_{\nu|\eta}\ \mathrm{Type}}{\Pi \vdash_\ell T \ \leq\ Y_{\nu|\eta}} \quad (\mathrm{REnd}_\leq)$$

$$\frac{X_{\alpha|\beta} \leq U \notin_2^\simeq \Pi \quad X_\alpha{=}T \in \mathrm{Left}(\Pi) \quad \Pi, X_{\alpha|\beta} \leq U \vdash_\ell T \uparrow \beta \ \leq\ U}{\Pi \vdash_\ell X_{\alpha|\beta} \ \leq\ U} \quad (\mathrm{LUnf}^2_\leq)$$

$$\frac{T \leq Y_{\nu|\eta} \notin_2^\simeq \Pi \quad \textit{for all } X_{\theta|\delta}.\ (T \neq X_{\theta|\delta}) \quad Y_\nu{=}U \in \mathrm{Right}(\Pi) \quad \Pi, T \leq Y_{\nu|\eta} \vdash_\ell T \ \leq\ U \uparrow \eta}{\Pi \vdash_\ell T \ \leq\ Y_{\nu|\eta}} \quad (\mathrm{RUnf}^2_\leq)$$

Appendix C: Encoding Pair and Bottom Types in Recursive Kernel Fun

In this appendix we show how to encode pair and bottom types in recursive Kernel Fun. For our aims, we do not need here to define types whose terms behave as pairs, or types with no terms; we only need types which reflect pair and bottom type *subtyping* behaviour.

Our encoding of pair types is very similar to that proposed by Cardelli et al. in [CMMS94], where pair types are encoded in terms of unbounded quantification and arrow types, in an extension of system F with subtyping. In that paper, if we assume A and B not containing pair types, pairs are encoded as follows:

$$A \times B \triangleq \forall t.(A \to B \to t) \to t$$

with t not free in A and B.

In [CMMS94], the encoding of bottom type is not considered. Here we propose an encoding that works for both pairs and bottom. More formally, we provide an encoding from types

$$A, B ::= \bot \mid \top \mid t \mid X \mid \mu X.\forall t \leq A.B \mid \mu X.A \to B \mid \mu X.A \times B$$

to types

$$T, U ::= \top \mid t \mid X \mid \mu X.\forall t \leq T.U \mid \mu X.T \to U$$

Hereafter, we use letters A, B to denote types with bottom and pairs.

We impose that the bound B in $\mu X.\forall t \leq B.A$ must not be $\bot$; this limitation preserves the antireflexivity of the system, and does not limit its expressive power, since a type $\mu X.\forall t \leq \bot.A$ just introduces a variable t that is equivalent to $\bot$.

Subtyping among these types is defined by rules $\Re^{\infty}$ together with the following two rules.

$$\frac{\Pi \vdash \bot, A \text{ Type}}{\Pi \vdash \bot \leq A} \quad (\bot_{\leq})$$

$$\frac{\begin{array}{l}\Pi' = (\Pi, (X{=}\mu X.A \times B,\ Y{=}\mu Y.A' \times B')) \\ \Pi' \vdash A \leq A' \quad \Pi' \vdash B \leq B'\end{array}}{\Pi \vdash \mu X.A \times B \leq \mu Y.A' \times B'} \quad (\times_{\leq})$$

To define the encoding, we first define *type negation* $\neg T$ as follows:

$$\neg T \triangleq \mu.(T \to \top)$$

where $\mu.U$ stands for a type $\mu Y.U$ where Y does not occur free in U.

It is easy to prove that:

$$() \vdash \neg T \leq \neg U \Leftrightarrow () \vdash U \leq T$$

and therefore $() \vdash \neg\neg T \leq \neg\neg U \Leftrightarrow () \vdash T \leq U$.

The encoding is defined by a pair of functions, external double negation $[\![\_]\!]$ and internal double negation $(\!|\_|\!)$. Informally, $[\![A]\!]$ doubly negates A and all its subterms, while $(\!|A|\!)$ only double-negates the subterms. The encoding is based on the fact that, since $\top \geq \neg T$ for any T, then $\neg\top \leq \neg\neg T$ for any T, hence $\neg\top$ is the bottom type in a set where any other type is doubly negated. However, the bound of a bounded type variable cannot be doubly negated at the outermost level.

Consider a comparison $\forall t{\leq}A.t \leq \forall t{\leq}A.A$. It gets translated as $\forall t{\leq}(\!|A|\!).\neg\neg t \leq \forall t{\leq}(\!|A|\!).[\![A]\!]$. The rule $\forall$ reduces this into $t{\leq}(\!|A|\!) \vdash \neg\neg t \leq [\![A]\!]$ which is then reduced to $t{\leq}(\!|A|\!) \vdash t \leq (\!|A|\!)$, and finally to $t{\leq}(\!|A|\!) \vdash (\!|A|\!) \leq (\!|A|\!)$, as desired. Here, it is crucial that the substitution of t with its bound does not add a new pair of negations around $(\!|A|\!)$. This is even clearer if we consider a situation like $v{\leq}w, u{\leq}v, t{\leq}u \vdash [\![t]\!] \leq [\![w]\!]$. It is essential that t is substituted by u and u by w with no double negation added in the process. For this reason, we use internal double negation for variable bounds. Internal double negation is well defined since such bounds are never equal to $\bot$. The bound regains the needed external double negation, in a sense, when it is substituted to a variable type t in the comparison, since t was doubly negated to begin with.

Formally, $[\![\_]\!]$ and $(\!|\_|\!)$ are defined by mutual recursion as follows. $[\![A]\!]$ is total, while $(\!|A|\!)$ is undefined on $\bot$. Internal double negation is defined on bi-environments as well.

$$\begin{array}{ll} [\![\bot]\!] \triangleq \neg\top & \\ [\![A]\!] \triangleq \neg\neg(\!|A|\!) & \textit{if } A \neq \bot \\ (\!|\bot|\!) & \textit{undefined} \end{array}$$

$$
\begin{aligned}
(\!|\top|\!) &\triangleq \top \\
(\!|t|\!) &\triangleq t \\
(\!|X|\!) &\triangleq X \\
(\!|\mu X.A \to B|\!) &\triangleq \mu X.[\![A]\!] \to [\![B]\!] \\
(\!|\mu X.A \times B|\!) &\triangleq \mu X.\forall s \leq [\![\top]\!].\mu.(\mu.[\![A]\!] \to \mu.([\![B]\!] \to s)) \to s \\
&\qquad \textit{where } s \textit{ is a fresh variable} \\
(\!|\mu X.\forall t \leq A.B|\!) &\triangleq \mu X.(\forall t \leq (\!|A|\!).[\![B]\!]) \\
(\!|\Pi, (t,u) \leq (A,B)|\!) &\triangleq (\!|\Pi|\!), (t,u) \leq ((\!|A|\!), (\!|B|\!)) \\
(\!|\Pi, (X = A, Y = B)|\!) &\triangleq (\!|\Pi|\!), (X = (\!|A|\!), Y = (\!|B|\!))
\end{aligned}
$$

This encoding enjoys the following property, where $\vdash_{\bot,\times}$ is the subtype relation of the system extended with $\bot$ and $\times$.

$$
\begin{aligned}
&\Pi \vdash_{\bot,\times} A \leq B \Leftrightarrow (\!|\Pi|\!) \vdash (\!|A|\!) \leq (\!|B|\!) \\
&\Pi \vdash_{\bot,\times} A \leq B \Leftrightarrow (\!|\Pi|\!) \vdash [\![A]\!] \leq [\![B]\!]
\end{aligned}
$$

We have first to prove the following inversion lemma.

Lemma 7.1 $(\!|\Pi|\!) \vdash (\!|A_1|\!) \leq (\!|B_1|\!)$ *implies that one of the following conditions holds.*

$A_1 = A,\ B_1 = \top,\ \Pi \vdash (\!|A|\!)$ *Type*

$A_1 = t,\ B_1 = u,\ (t,u) \leq ((\!|A'|\!), (\!|B'|\!)) \in (\!|\Pi|\!)$

$A_1 = t,\ (t,u') \leq ((\!|A'|\!), (\!|B'|\!)) \in (\!|\Pi|\!)$,
for all X. $((\!|B_1|\!) \neq X), (\!|B_1|\!) \neq \top, (\!|B_1|\!) \neq u,\ (\!|\Pi|\!) \vdash (\!|A'|\!) \leq (\!|B_1|\!)$

$A_1 = \mu X.A \to B,\ B_1 = \mu Y.A' \to B'$,
$\Pi' = (\!|\Pi, (X = \mu X.A \to B,\ Y = \mu Y.A' \to B')|\!)$,
$\mathrm{Swap}(\Pi') \vdash (\!|A'|\!) \leq (\!|A|\!),\ \Pi' \vdash (\!|B|\!) \leq (\!|B'|\!)$

$$A_1 = \mu X.A \times B,\ B_1 = \mu Y.A' \times B',$$
$$\Pi' = (\!|\Pi, (X{=}\mu X.A \times B,\ Y{=}\mu Y.A' \times B')|\!),$$
$$\Pi' \vdash (\!|A|\!) \leq (\!|A'|\!),\ \Pi' \vdash (\!|B|\!) \leq (\!|B'|\!)$$

$$A_1 = \mu X.\forall t{\leq}A.B,\ B_1 = \mu Y.\forall t'{\leq}A'.B',$$
$$\Pi' = (\!|\Pi, (X{=}\mu X.\forall t{\leq}A.B,\ Y{=}\mu Y.\forall t'{\leq}A'.B')|\!),$$
$$\Pi' \vdash (\!|A|\!) \mathrel{\mathring{=}} (\!|A'|\!),\ \Pi', (t,t'){\leq}((\!|A|\!),(\!|A'|\!)) \vdash (\!|B|\!) \mathrel{\mathring{=}} (\!|B'|\!)$$

$$A_1 = X,\ B_1 = B,\ X{=}(\!|A'|\!) \in \mathrm{Left}(\Pi),\ (\!|\Pi|\!) \vdash\ (\!|A'|\!)\uparrow{\leq}\ (\!|B|\!)$$

$$A_1 = A,\ B_1 = Y,\ \forall X.\ A_1 \neq X,\ Y{=}(\!|B'|\!) \in \mathrm{Right}(\Pi),$$
$$(\!|\Pi|\!) \vdash (\!|A|\!)\ \leq\ (\!|B'|\!)\uparrow$$

Proof. For any A, if its outer constructor is different from $\times$, the outer constructor of $(\!|A|\!)$ is the same as that of A, hence $(\!|A|\!) \leq (\!|B|\!)$ can only be proved by using the rules that correspond to their outer constructor. If one among A and B is a product, then it is mapped to a μ-$\forall$ type, which may be compared, in principle, to the image of a μ-$\forall$ type. However, $\mu X.A \times B$ types are mapped into μ-$\forall$ types where the upper bound for the variable is $[\![\top]\!]$, that is $\neg\neg\top$. Types $\mu X.\forall t \leq A.B$, instead, are mapped into μ-$\forall$ types where the corresponding bound is $(\!|A|\!)$, which is always different from $[\![\top]\!]$. This ensures that it is impossible to prove that $[\![\mu X.\forall t \leq A.B]\!] \leq (\!|\mu Y.A' \times B'|\!)$, and vice versa. The rest of the proof is trivial. □

Lemma 7.2 $\quad (\!|\Pi|\!) \vdash [\![A]\!] \leq [\![B]\!] \ \Leftrightarrow ((\!|\Pi|\!) \vdash (\!|A|\!) \leq (\!|B|\!) \vee A = \bot)$

We can now prove the basic theorems.

Theorem 7.3 $\quad (\!|\Pi|\!) \vdash [\![A]\!] \leq [\![B]\!] \Rightarrow \Pi \vdash_{\bot,\times} A \leq B$

Proof. We give a coinductive (aka lazy) proof of the existence of a function $\mathbf{T}$ that transforms any proof, finite or infinite, of $(\!|\Pi|\!) \vdash [\![A]\!] \leq [\![B]\!]$ into a proof of $\Pi \vdash_{\bot,\times} A \leq B$. If $A = \bot$ the thesis is immediate. Otherwise, by Lemma 7.2, we have that $(\!|\Pi|\!) \vdash (\!|A|\!) \leq (\!|B|\!)$, hence we only have to examine the cases of Lemma 7.1, and each case allows us to build a piece of the desired proof. Assume, for example, that we are in case $X \leq B$. Lemma 7.1 guarantees the existence of a proof of $(\!|\Pi|\!) \vdash\ (\!|A'|\!)\uparrow{\leq}\ (\!|B|\!)$. Hence, Lemma 7.2 guarantees the existence of a proof of $(\!|\Pi|\!) \vdash\ [\![A']\!]\uparrow{\leq}\ [\![B]\!]$, hence of $(\!|\Pi|\!) \vdash\ [\![A'\uparrow]\!]\ \leq\ [\![B]\!]$. Hence, $\mathbf{T}$ can return a proof for $\Pi \vdash_{\bot,\times} X \leq B$ that is formed by an instance of the μ rule applied to the result of applying $\mathbf{T}$ to a proof of $(\!|\Pi|\!) \vdash\ [\![A'\uparrow]\!]\ \leq\ [\![B]\!]$. □

Theorem 7.4 $\quad \Pi \vdash_{\bot,\times} A \leq B \Rightarrow (\!|\Pi|\!) \vdash [\![A]\!] \leq [\![B]\!]$

Proof. We give a coinductive definition of a function that transforms any proof

of $\Pi \vdash_{\perp,\times} A \leq B$ into a proof of $(\!|\Pi|\!) \vdash \llbracket A \rrbracket \leq \llbracket B \rrbracket$. $\square$