Subtyping Recursive Types

Roberto M. Amadio[1]
LIENS, Ecole Normale Supérieure, Paris

Luca Cardelli
DEC, Systems Research Center

Abstract

Subtyping is an inclusion relation between types that is present to some degree in many programming languages. Subtyping is especially important in object-oriented languages, where it is crucial for understanding the much more complex notions of inheritance and subclassing. Recursive types are also present in most languages; these types are supposed to *unfold* recursively to match other types.

In this paper we investigate the interaction of unrestricted recursive types with subtyping, which we refer to as recursive subtyping, in the context of structural type matching. This interaction is present in some modern languages such as Modula-3 [Cardelli Donahue Jordan Kalsow Nelson 89] and Quest [Cardelli 89].

More precisely, we relate various notions of type equivalence ($\alpha=\beta$) and subtyping ($\alpha<\beta$) in a simply typed λ-calculus with subtyping and recursive types. These relations are induced by:

A collection of *models*:	$\alpha=_M\beta$,	$\alpha<_M\beta$
An *ordering on infinite trees*:	$\alpha=_T\beta$,	$\alpha<_T\beta$
An *algorithm*:	$\alpha=_A\beta$,	$\alpha<_A\beta$
A collection of *typing rules*:	$\alpha=_R\beta$,	$\alpha<_R\beta$

The mathematical content of the paper consists mainly in analysing the relationships between these notions. We show, among other properties:

$$\alpha=_T\beta \Leftrightarrow \alpha=_R\beta \Leftrightarrow \alpha=_A\beta \Rightarrow \alpha=_M\beta$$
$$\alpha<_T\beta \Leftrightarrow \alpha<_R\beta \Leftrightarrow \alpha<_A\beta \Rightarrow \alpha<_M\beta,$$

We also prove a restricted form of completeness with respect to the model.

Moreover, we show that to every pair of types in subtype relation we can associate a term whose denotation is the uniquely determined coercion map between the two types. We then derive an algorithm that, given a term with implicit coercions, can infer its minimum type whenever possible.

[1]On leave from Dipartimento di Informatica, Università di Pisa. This author's work has been supported in part by Digital Equipment Corporation and in part by the Stanford-CNR Collaboration Project.

PROC. POPL '91

1. Introduction

A *type*, as normally intended in programming languages, is a collection of values sharing a common structure or shape. Examples of *basic* types are: Unit, the trivial type containing a single element, and Int, the collection of integer numbers. Examples of *structured* types are: Int→Int, the functions from integers to integers; Int×Int, the pairs of two integers; and Unit+Int, the *disjoint union* of Unit and Int.

A *recursive type* is a type that satisfies a *recursive type equation*, such as the following equations for binary trees with integer leaves and for lists of integers (there are also useful examples of recursion involving function spaces, which are typical of the object-oriented style of programming):

Tree = Int+(Tree×Tree)
List = Unit+(Int×List)

Note that these are not definitions: these are equational properties that Tree and List must satisfy.

We use a term such as $\mu t.\text{Unit}+(\text{Int}\times t)$ (*the* type t that is equal to Unit+(Int×t)) to indicate the *canonical* solution of a type equation, which we can show to exist. To say that $L \triangleq \mu t.\text{Unit}+(\text{Int}\times t)$ (where $\triangleq$ means *equal by definition*) is the solution of the List equation, means that L must provably satisfy the equation. This is achieved via an unfolding rule:

$$\mu t.\alpha = [\mu t.\alpha/t]\alpha$$

meaning that $\mu t.\alpha$ is equal to α where we replace t by $\mu t.\alpha$ itself. In our example we have $L = \mu t.\text{Unit}+(\text{Int}\times t) = [L/t](\text{Unit}+(\text{Int}\times t)) = \text{Unit}+(\text{Int}\times L)$, which is the equation we expected to hold.

If types are collections of values, *subtypes* should be subcollections. For example, we can introduce two new basic types $\bot$ (*bottom*), the collection containing only the divergent computation, and $\top$ (*top*), the collection of all values. Then $\bot$ should be a subtype of every type, and every type should be a subtype of $\top$. We write these relations as $\bot<\alpha$ and $\alpha<\top$.

Function spaces $\alpha\to\beta$ have a subtyping rule that is *antimonotonic* in the first argument. That is:

$$\alpha\to\beta < \alpha'\to\beta' \text{ if } \alpha' < \alpha \text{ and } \beta < \beta'.$$

Adequate subtyping rules can be found for all the other type constructions we may have. For example, for products we have $\alpha\times\beta < \alpha'\times\beta'$ if $\alpha < \alpha'$ and $\beta < \beta'$. Similarly, we have $\alpha+\beta < \alpha'+\beta'$ if $\alpha < \alpha'$ and $\beta < \beta'$. To deal with recursive types, however, we must first talk about type equivalence.

To check whether two recursive types $\mu s.\alpha'$ and $\mu t.\beta'$ are equivalent, we could assume s=t, and attempt to prove $\alpha'=\beta'$ under this assumption. Unfortunately, the following types:

$$\alpha \triangleq \mu s.\mathrm{Int}\to s \qquad \beta \triangleq \mu t.\mathrm{Int}\to\mathrm{Int}\to t$$

expand both to $\mathrm{Int}\to\mathrm{Int}\to\mathrm{Int}\to\mathrm{Int}\to\ldots$, but the assumption s=t does not allow us to show $\mathrm{Int}\to s = \mathrm{Int}\to\mathrm{Int}\to t$; we get stuck on the question of whether $s = \mathrm{Int}\to t$.

Another attempt might involve expanding the μ's, but by unfolding alone we can get only:

$$\begin{aligned}\alpha &= \mu s.\mathrm{Int}\to t = \mathrm{Int}\to(\mu s.\mathrm{Int}\to t)\\ &= \mathrm{Int}\to\mathrm{Int}\to(\mu s.\mathrm{Int}\to t) = \mathrm{Int}\to\mathrm{Int}\to\alpha\\ \beta &= \mu t.\mathrm{Int}\to\mathrm{Int}\to t = \mathrm{Int}\to\mathrm{Int}\to(\mu t.\mathrm{Int}\to\mathrm{Int}\to t)\\ &= \mathrm{Int}\to\mathrm{Int}\to\beta\end{aligned}$$

which, after unfolding, leads us back to the original problem of determining whether $\alpha=\beta$.

However, in the process above we have found a single context $C[X] \triangleq \mathrm{Int}\to\mathrm{Int}\to X$ such that $\alpha = C[\alpha]$ and $\beta = C[\beta]$; i.e., both α and β are fixpoints of C. We shall be able to show that all the non-trivial (formally, *contractive*) type contexts C[X] have *unique* fixpoints over infinite trees. Hence the necessary rule for determining type equality can be formulated as follows:

$$\alpha = C[\alpha] \wedge \beta = C[\beta] \wedge C \text{ contractive} \Rightarrow \alpha = \beta \quad (1)$$

It remains to be shown how to generate these contexts for any two types that expand to equal infinite trees. This can be done via an algorithm, and in fact a complete one.

What is, then, subtyping for recursive types? The intuition is that two recursive types α and β are in subtype relation if their *infinite* unfoldings are in this relation, in an appropriate sense.

We'll see that this informal idea can be axiomatized via the following (finitary) rule:

$$(s<t \Rightarrow \alpha < \beta) \Rightarrow \mu s.\alpha < \mu t.\beta \qquad (2)$$

where s occurs only in α, and t occurs only in β. That is, if by assuming s<t we can verify $\alpha < \beta$, then we can deduce the inclusion of the recursive types $\mu s.\alpha < \mu t.\beta$. For example, if we have Nat<Int, $\mathrm{NatList} \triangleq \mu s.\mathrm{Unit}+(\mathrm{Nat}\times s)$, and $\mathrm{IntList} \triangleq \mu t.\mathrm{Unit}+(\mathrm{Int}\times t)$, then we can safely deduce NatList<IntList from (2).

In general, rule (2) needs to interact with rule (1). Suppose we want to check, e.g., $\gamma<\delta$, where:

$$\gamma \triangleq \mu s.\mathrm{Int}\to s \qquad \delta \triangleq \mu t.\mathrm{Nat}\to\mathrm{Nat}\to t$$

Attempts to unfold γ and δ fall into the same difficulties as before. This problem is solved by a reduction to an equality problem, which we solve by rule (1), plus rule (2). That is, we first show that $\delta' \triangleq \mu t.\mathrm{Nat}\to t = \mu t.\mathrm{Nat}\to\mathrm{Nat}\to t \equiv \delta$. After that, we can use rule (2) to show $\gamma<\delta'$, and hence $\gamma<\delta$.

2. Calculus and Tree Ordering

We consider a simply typed λ-calculus with recursive types and two ground types $\bot$ (bottom) and $\top$ (top); the latter play the roles of least and greatest elements in the subtype relation.[3]

2.1 Types and Terms

In an informal BNF notation, *types* are defined as follows, (t,s,... are type variables):

$$\alpha ::= t \mid \bot \mid \top \mid (\alpha\to\beta) \mid (\mu t.\alpha)$$

Terms are denoted with M, N, ... ; the following rules establish when a term M has type α (M:α).

(assmp)	$x^\alpha : \alpha$
($\to$I)	$M:\beta \Rightarrow (\lambda x^\alpha.M) : \alpha\to\beta$
($\to$E)	$M:\alpha\to\beta,\ N:\alpha \Rightarrow (MN):\beta$
(fold)	$M:[\mu t.\alpha/t]\alpha \Rightarrow (\mathrm{fold}_{\mu t.\alpha} M) : \mu t.\alpha$
(unfold)	$M:\mu t.\alpha \Rightarrow (\mathrm{unfold}_{\mu t.\alpha} M) : [\mu t.\alpha/t]\alpha$

2.2 Subtyping Non-recursive Types

There is a well-established theory of subtyping for the non-recursive types (i.e. types non containing μ's). Basic motivations can be found, e.g., in [Cardelli 88]. The rule are:

($\bot$)	$\bot < \alpha$	(var)	$t < t$
($\top$)	$\alpha < \top$	($\to$)	$\alpha'<\alpha,\ \beta<\beta' \Rightarrow \alpha\to\beta<\alpha'\to\beta'$

These rules induce a partial order on the collection of non-recursive types.

2.3 Tree Expansions and Approximations

As we have seen in the introduction, simple unfolding does not induce a sufficiently strong type equality. We have also seen that it is natural to consider types with the same infinite expansions as equivalent. Infinite expansion can be rephrased as an *approximation property* such that the semantics of a type is entirely determined by the semantics of its finite syntactic approximations.

We start formalizing this idea by explaining how to associate a finitely branching, labeled, regular tree with any recursive type. Paths in a tree are represented by finite sequences of natural numbers $\pi,\sigma\in\omega^*$, with $\pi\sigma$ for concatenation and *nil* as the empty sequence. Nodes in a tree are labeled by a ranked alphabet $L = \{\bot^0, \top^0, \to^2\} \cup \{t^0 \mid t \text{ is a type variable}\}$, where the superscripts indicate arity. A tree $A\in\omega^*\rightharpoonup L$ is a partial function from

[3] Conventions: $\triangleq$ stands for equality by definition; $\equiv$ for abbreviation or syntactic identity; $\vdash$ precedes a judgment that is provable in a certain formal system; $\supset$ denotes thelinguistic implication; $\Rightarrow$ is metalinguistic implication; [U/x]V denotes the substitution of U for x in V.

(paths) ω^* into (node labels) L, whose domain is non-empty and prefix-closed, and such that each node has a number of siblings equal to the rank of its label.

We call Tree(L) the set of such trees, with $\text{Tree}_{\text{fin}}(L)$ for the finite ones. We now define a function $T \in \text{Type} \to \text{Tree}(L)$ by induction on the pair $(|\pi|, \alpha)$ where $|\pi|$ is the length of the path π in the tree associated with α. The following schemas are meant to suggest the correct definition:

$$T\bot \triangleq \bot \qquad T\top \triangleq \top \qquad Tt \triangleq t$$

$$T\alpha\to\beta \triangleq \begin{array}{c} \to \\ / \ \backslash \\ T\alpha \ \ T\beta \end{array}$$

$$T\mu t.\alpha \triangleq \begin{cases} \bot & \text{if } \alpha \equiv \mu t_1. \ldots \mu t_n.\, t \ \ (t_i \neq t,\ i \in 1..n,\ n \geq 0) \\ T[\mu t.\alpha/t]\alpha & \text{otherwise} \end{cases}$$

Remarks

T induces a bijection between $\text{Tree}_{\text{fin}}(L)$ and non-recursive types. We denote with T^{-1} its inverse. This bijection is sometimes left implicit.

Tree(L) is a complete metric space with respect to the usual metric on trees [Arnold Nivat 80]; in fact, it is the completion of $\text{Tree}_{\text{fin}}(L)$. Hence, Tree(L) has unique solutions for recursive contractive tree equations.

Given types α, β, the problem of deciding if $T\alpha = T\beta$ is reducible to the problem of the equivalence of deterministic finite-state automata [Courcelle 83].

We now extend the partial order on finite trees (induced by 2.2) to infinite trees, by introducing a notion of finite approximation. We define a family of functions: $\{\,|_k \in \text{Tree}(L) \to \text{Tree}_{\text{fin}}(L)\}_{k\in\omega}$. Given $A \in \text{Tree}(L)$ its *cut at the k-th level* ($A|_k$) is defined as follows ($\uparrow$ and $\downarrow$ indicate divergence and convergence for partial functions):

$$A|_k(\pi) \triangleq \begin{cases} \uparrow & \text{if } |\pi| > k \\ A(\pi) & \text{if } |\pi| < k, \text{ or } |\pi| = k \text{ and } A(\pi)\uparrow \\ \bot & \text{if } |\pi| = k,\ A(\pi)\downarrow, \text{ and } \pi \text{ is positive in A} \\ \top & \text{if } |\pi| = k,\ A(\pi)\downarrow, \text{ and } \pi \text{ is negative in A} \end{cases}$$

where we say that π is positive (negative) in A if along the path π from the root we select the left sibling of a node labeled $\to$ an even (odd) number of times.

We can then extend this definition to types:

$$\alpha|_k \ \triangleq \ T^{-1}((T\alpha)|_k) \qquad \text{(a non-recursive type).}$$

Definition 2.3.1 *(tree ordering)*

For $A,B \in \text{Tree}_{\text{fin}}(L)$: $A <_{\text{fin}} B \iff T^{-1}A < T^{-1}B$ (2.2)
For $A,B \in \text{Tree}(L)$: $A <_\infty B \iff \forall k.\, (A|_k <_{\text{fin}} B|_k)$
For $\alpha,\beta \in \text{Type}$: $\alpha <_T \beta \iff T\alpha <_\infty T\beta$

Note that $<_\infty$ is a partial order on Tree(L), and $\alpha <_T \beta$ is a preorder on recursive types such that $\forall k.\, \alpha|_k <_T \alpha$. We can now show, e.g.: $\mu t.\top\to t <_T \mu t.\bot\to(\bot\to t)$.

3. An Algorithm

In this section we show that the tree ordering we have defined on types (2.3.1) can be decided by a rather natural modification of the algorithm that tests directly (that is, without reduction to a minimal form) the tree equivalence of two types.

3.1 Canonical Forms

The first step towards formalizing the algorithm is to introduce canonical forms for types and systems of equations.

Next we define maps that associate with such representations the corresponding regular trees. Moreover, we give effective ways of going back and forth among the representations while preserving the represented tree.

Proviso

In order to have a simple correspondence between recursive types and systems of regular equations, we assume that all variables, both bound and free, in the types $\alpha_1, \ldots, \alpha_n$ under consideration are distinct. When a type is unfolded, the necessary renaming of bound variables must be performed. For example, $(\mu t.t\to s)\to(\mu s.t\to s)$ should be rewritten as $(\mu v.v\to s)\to(\mu r.t\to r)$.

3.1.1 Recursive Types in Canonical Form

Henceforth, Tp denotes the collection of non-recursive types, and μTp denotes the collection of *recursive types in canonical form*, defined as follows:

$$\alpha \ ::= \ \bot \mid \top \mid t \mid \alpha\to\beta \mid \mu t.\alpha\to\beta$$

where in the case $\mu t.\alpha\to\beta$, t must occur free in $\alpha\to\beta$. Hence the body of a μ in canonical form must immediately start with an $\to$; in particular, it cannot be another μ. The introduction of μTp simplifies the case analysis in the following proofs.

It is not difficult to check that for every type α there is a type β in canonical form such that $T\alpha = T\beta$. The crucial observation is that $T\mu t.\mu s.\gamma[t,s] = T\mu v.\gamma[v,v]$. See also 4.1.3 for a proof of this fact that uses the rules for type equivalence.

3.1.2 Regular System of Equations in Canonical Form

Systems of regular equations are a well-known tool for representing regular trees (see for example [Courcelle 83]).

For our purposes a *regular system of equations in canonical form* is an element of Tenv, that is, a finite association of distinct type variables (members of Tvar) with types in a specific form:

$$\text{Tenv} \triangleq$$
$$\{\, \varepsilon \in \text{Tvar} \rightharpoonup \text{Tp} \mid \text{Dom}(\varepsilon) \text{ is finite and } \forall t \in \text{Dom}(\varepsilon).\ \varepsilon(t) ::= \bot \mid \top \mid s \mid t_1 \to t_2$$
$$\text{where } s \notin \text{Dom}(\varepsilon) \text{ and } t_1, t_2 \in \text{Dom}(\varepsilon) \,\}$$

A pair $(\alpha, \varepsilon) \in \text{Tp} \times \text{Tenv}$ represents the following *system of regular equations* (not necessarily in canonical form):

$t_\alpha = \alpha$ (t_α a fresh variable)
$s = \varepsilon(s)$ for each $s \in Dom(\varepsilon)$

It is important to observe that this system defines a *contractive functional* $(G_0,..., G_n)$ over $Tree(L)^{n+1}$ where $n = |Dom(\varepsilon)|$, $Dom(\varepsilon) = \{s_1, ..., s_n\}$ and

$G_0(A_0, ..., A_n) \triangleq [A_0/t_\alpha, ..., A_n/s_n]\alpha$
$G_i(A_0, ..., A_n) \triangleq [A_0/t_\alpha, ..., A_n/s_n]\varepsilon(s_i)$ $(1 \le i \le n)$

$Reach(\alpha,\varepsilon)$ denotes the variables reachable from the free variables in α by applying the equations in ε.

3.1.3 Definition *(solution of a system)*
We denote with $Sol(\alpha, \varepsilon)$ the first component B_0 of the solution $(B_0, ..., B_n)$ in $Tree(L)^{n+1}$ of the system associated with (α, ε).

Remark
Given a system of regular equations in canonical form, it is possible to minimize the number of variables by a procedure that is analogous to the one for minimizing the number of states in a deterministic finite-state automaton. This immediately provides an algorithm for deciding the equality of the trees represented by two regular systems of equations in canonical form.

3.1.4 Proposition *(From rec. types to regular systems)*
There is a pair of maps:
$* \in Type \to Tvar$, $E \in Type \to Tenv$, such that:
$\forall \alpha \in Type.\ T\alpha = Sol(\alpha^*, E\alpha)$

Proof
It is enough to prove the result for every term in μTp. Then the lemma follows by 3.1.1. We now define $(*, E)$ by induction on the structure of $\gamma \in \mu Tp$.

Cases $\gamma \equiv t$, $\gamma \equiv \bot$, and $\gamma \equiv \top$. Take $\gamma^* \triangleq s$ and $E\gamma \triangleq \{s = \gamma\}$.

Case $\gamma \equiv \alpha \to \beta$.
We denote with $p[t_1...t_n]$ for $p \in L \setminus \{t_1...t_n\}$ a type in Tp of the form $p(u_1..u_{\#p})$, where #p is the arity of p, and $\{u_1..u_{\#p}\} \subseteq \{t_1...t_n\}$.
Assume, by induction hypothesis, that $E\alpha = \{t_{i+1} = p_i[t_2...t_{n+1}] \mid i \in 1..n\}$, $\alpha^* = t_2$, $E\beta = \{t_{n+1+j} = q_j[t_{n+2}...t_{n+m+1}] \mid j \in 1..m\}$, and $\beta^* = t_{n+2}$. Then $E\gamma$ is the following system and $\gamma^* \triangleq t_1$:

$t_1 = r_1[t_1...t_{n+m+1}] \equiv t_2 \to t_{n+2}$,
$t_2 = r_2[t_1...t_{n+m+1}] \equiv p_1[t_2...t_{n+1}]$ ...
$t_{n+1} = r_{n+1}[t_1...t_{n+m+1}] \equiv p_n[t_2...t_{n+1}]$
$t_{n+2} = r_{n+2}[t_1...t_{n+m+1}] \equiv q_1[t_{n+2}...t_{n+m+1}]$ ...
$t_{n+m+1} = r_{n+m+1}[t_1...t_{n+m+1}] \equiv q_m[t_{n+2}...t_{n+m+1}]$

Case $\gamma \equiv \mu t.\alpha \to \beta$.
Let $\gamma' = [\gamma/t]\alpha \to [\gamma/t]\beta$ (of course $T\gamma' = T\gamma$). As in the previous case, assume $E\alpha = \{t_{i+1} = p_i[t_2...t_{n+1}] \mid i \in 1..n\}$, $\alpha^* = t_2$, $E\beta = \{t_{n+1+j} = q_j[t_{n+2}...t_{n+m+1}] \mid j \in 1..m\}$ and $\beta^* = t_{n+2}$. Then $E\gamma$ is the following system and $\gamma^* \triangleq t_1$:

$t_1 = r_1[t_1...t_{n+m+1}] \equiv t_2 \to t_{n+2}$
$t_2 = r_2[t_1...t_{n+m+1}] \equiv t_2 \to t_{n+2} \mid p_1[t_2...t_{n+1}]$ ...
$t_{n+1} = r_{n+1}[t_1...t_{n+m+1}] \equiv t_2 \to t_{n+2} \mid p_n[t_2...t_{n+1}]$
$t_{n+2} = r_{n+2}[t_1...t_{n+m+1}] \equiv t_2 \to t_{n+2} \mid q_1[t_{n+2}...t_{n+m+1}]$ ...
$t_{n+m+1} = r_{n+m+1}[t_1...t_{n+m+1}] \equiv t_2 \to t_{n+2} \mid q_m[t_{n+2}...t_{n+m+1}]$

By $t_2 \to t_{n+2} \mid p_i[t_2...t_{n+1}]$ we denote $t_2 \to t_{n+2}$ if $p_i \equiv t$, and $p_i[t_2...t_{n+1}]$ otherwise. Analogously, $t_2 \to t_{n+2} \mid q_j[t_{n+2}...t_{n+m+1}]$ denotes $t_2 \to t_{n+2}$ if $q_j \equiv t$, and $q_j[t_{n+2}...t_{n+m+1}]$ otherwise. Next proceed by induction on $(|\pi|, \gamma)$ to prove $T\alpha(\pi) = Sol(\alpha^*, E\alpha)(\pi)$. The only difficulty arises for $\gamma \equiv \mu t.\alpha \to \beta$. In order to apply the induction hypothesis one needs to show, for example, $Sol(t_2, E\gamma) = Sol([\gamma/t]\alpha^*, E[\gamma/t]\alpha)$. □

3.1.5 Definition *(From regular systems to rec. types)*
We define a function $\langle -, - \rangle : Tp \times Tenv \to \mu Tp$ by induction on $(|Dom(\varepsilon)|, \alpha)$:

$\langle \bot, \varepsilon \rangle \triangleq \bot$ $\langle \top, \varepsilon \rangle \triangleq \top$ $\langle t, \varepsilon \rangle \triangleq t$ if $t \notin Dom(\varepsilon)$
$\langle \alpha \to \beta, \varepsilon \rangle \triangleq \langle \alpha, \varepsilon \rangle \to \langle \beta, \varepsilon \rangle$
$\langle t, \varepsilon \rangle \triangleq \mu t.\langle \varepsilon(t), \varepsilon \setminus t \rangle$ if $t \in Dom(\varepsilon)$

where $\varepsilon \setminus t$ is like ε except that it is undefined on t.

3.1.6 Proposition *(More on commuting translations)*
(1) For any system of equations, the first component of the solution coincides with the tree expansion of the associated recursive type:
$\forall (\alpha, \varepsilon) \in Tp \times Tenv.\ Sol(\alpha, \varepsilon) = T\langle \alpha, \varepsilon \rangle$
(2) The map $\langle , \rangle$ satisfies the conditions:
1. $\langle \bot, \varepsilon \rangle = \bot$
2. $\langle \top, \varepsilon \rangle = \top$
3. $\langle t, \varepsilon \rangle = t$ if $t \notin Dom(\varepsilon)$
4. $T\langle t, \varepsilon \rangle = T\langle \varepsilon(t), \varepsilon \rangle$ if $t \in Dom(\varepsilon)$
5. $T\langle \alpha \to \beta, \varepsilon \rangle = T(\langle \alpha, \varepsilon \rangle \to \langle \beta, \varepsilon \rangle)$
6. $T\langle \alpha^*, E\alpha \rangle = T\alpha$

Proof
(1) Show by induction on $(|\pi|, \alpha)$ that $T\langle \alpha, \varepsilon \rangle(\pi) = Sol(\alpha, \varepsilon)(\pi)$. The interesting case arises when $\alpha \equiv t$, $t \in Dom(\varepsilon)$. Then, $Sol(t, \varepsilon) = Sol(\varepsilon(t), \varepsilon) = Sol(t_1 \to t_2, \varepsilon)$, where $\varepsilon(t) = t_1 \to t_2$. On the other hand, $T\langle t, \varepsilon \rangle = T\mu t.\langle \varepsilon(t), \varepsilon \setminus t \rangle = T[\langle t, \varepsilon \rangle / t]\langle t_1 \to t_2, \varepsilon \setminus t \rangle$. In order to apply the induction hypothesis one needs to prove $T([\langle t, \varepsilon \rangle / t]\langle t_i, \varepsilon \setminus t \rangle) = T\langle t_i, \varepsilon \rangle$ $(i = 1,2)$.

(2) Conditions 1, 2, 3, 5 follow by definition. Condition 4 follows from $Sol(t, \varepsilon) = Sol(\varepsilon(t), \varepsilon)$ and part (1). Condition 6 follows from prop. 3.1.4 and part (1): $T\alpha = Sol(\alpha^*, E\alpha) = T\langle \alpha^*, E\alpha \rangle$. □

3.2 Computational Rules
We specify the algorithm as a set of rules for judgments (corresponding to procedure calls) of the form $\Sigma, \varepsilon \supset \alpha < \beta$, where $\Sigma = \{t_1 < s_1, ..., t_n < s_n\}$. The algorithm execution can be recovered by reading the rules backwards from the conclusions to the assumptions. In the following, t,s,r,u denote arbitrary variables; a,b denote variables not in the domain of ε; Σ is a finite set of subtyping assumptions on pairs of type variables; and $\alpha, \beta \in Tp$.

For $\Sigma = \{t_1 < s_1, ..., t_n < s_n\}$ let:
$Vars(\Sigma) \triangleq \{t_1, s_1, ..., t_n, s_n\}$
$\Sigma \# \varepsilon \Leftrightarrow Vars(\Sigma) \cap Dom(\varepsilon) = \varnothing$

A judgment $\Sigma, \varepsilon \supset \alpha < \beta$ satisfies the *initiality condition* (or equivalently, is an *initial goal*) iff $\alpha \equiv t$, $\beta \equiv s$, ε can be decomposed in $\varepsilon_1 \cup \varepsilon_2$ so that $t \in Dom(\varepsilon_1)$ and

$s \in Dom(\varepsilon_2)$, $Dom(\varepsilon_1) \cap Dom(\varepsilon_2) = \emptyset$, and $\Sigma \# \varepsilon$.

Observe that, by the shape of canonical systems, the expansions of variables according to ε is always synchronized; that is, in $\Sigma,\varepsilon \supset \alpha < \beta$ we never have a situation where α is a variable in $Dom(\varepsilon)$ and β is not, or vice versa.

$(assmp_A)$ $\Sigma,\varepsilon \supset t<s$ if $t<s \in \Sigma$

$(\bot_A)$ $\Sigma,\varepsilon \supset \bot<\beta$

$(\top_A)$ $\Sigma,\varepsilon \supset \alpha<\top$

(var_A) $\Sigma,\varepsilon \supset a<a$

$(\rightarrow_A)$ $\Sigma,\varepsilon \supset \alpha'<\alpha,\ \Sigma,\varepsilon \supset \beta<\beta' \Rightarrow \Sigma,\varepsilon \supset \alpha \rightarrow \beta < \alpha' \rightarrow \beta'$

(μ_A) $\Sigma \cup \{t<s\},\varepsilon \supset \varepsilon(t) < \varepsilon(s) \Rightarrow \Sigma,\varepsilon \supset t < s$ if $t,s \in Dom(\varepsilon)$

Note also that there are two conceptually distinct uses of the rule $(assmp_A)$: one for the initial assumptions contained in Σ, and one for the assumptions inserted during the computation.

If one desires to treat more general systems of equations, then it might be necessary to introduce other μ-rules that take into account situations in which just an ε-expansion on the left (or the right) is needed. In these cases we would have rules like:

$(assmp'_A)$ $\Sigma,\varepsilon \supset \alpha<\beta$ if $\alpha<\beta \in \Sigma$

(μ_{lA}) $\Sigma \cup \{t<\alpha' \rightarrow \beta'\},\varepsilon \supset \varepsilon(t)<\alpha' \rightarrow \beta' \Rightarrow \Sigma,\varepsilon \supset t<\alpha' \rightarrow \beta'$ if $t \in Dom(\varepsilon)$

(μ_{rA}) $\Sigma \cup \{\alpha' \rightarrow \beta' < s\},\varepsilon \supset \alpha' \rightarrow \beta' < \varepsilon(s) \Rightarrow \Sigma,\varepsilon \supset \alpha' \rightarrow \beta' < s$ if $s \in Dom(\varepsilon)$.

3.2.1 Generating the Execution Tree

Given a goal $\Sigma,\varepsilon \supset t<s$, the algorithm consists in applying the inference rules backwards, generating subgoals in the cases $(\rightarrow_A)$ and (μ_A). This process is completely determined once we establish that $(assmp_A)$ has priority over the other rules and $(\bot_A)$ has priority over $(\top_A)$.

A tree of goals built this way is called an *execution tree*. If no rules are applicable to a certain subgoal, that branch of the execution tree is abandoned, and execution is resumed at the next subgoal, until all subgoals are exhausted.

3.2.2 Termination

The execution tree is always *finite*. Observe that if $t < s$ is the assumption that we add to Σ, then t and s are type variables in $Dom(\varepsilon)$. Also observe that the $(\rightarrow)$ rule shrinks the size of the current goal by replacing it with subexpressions of the goal, and that each application of a μ-rule enlarges Σ.

The bound on the depth of the execution tree for $\alpha <_A \beta$ is of the order of the product of the sizes of the two systems $E\alpha$, $E\beta$.

3.2.3 Algorithm Ordering

An execution tree *succeeds* if all the leaves correspond to an application of one of the rules $(assmp_A)$, $(\bot_A)$, $(\top_A)$, and (var_A). Dually, it *fails* if at least one leaf is an unfulfilled goal (no rule can be applied).

We write $\vdash_A \Sigma,\varepsilon \supset t<s$ iff $\Sigma,\varepsilon \supset t<s$ is an initial goal (3.2) and the corresponding execution tree succeeds.

Given recursive types α,β we write:

$$\alpha <_A \beta \ \Leftrightarrow\ \vdash_A \emptyset, E\alpha \cup E\beta \supset \alpha^* < \beta^*$$

For testing type equality, we can define:

$$\alpha =_A \beta \ \Leftrightarrow\ \alpha <_A \beta \wedge \beta <_A \alpha$$

Alternatively, we could directly define a (more efficient) type equality algorithm, along the same lines as the subtyping algorithm.

3.3 Soundness and Completeness of the Algorithm

We now show that the algorithm is sound and complete with respect to infinite trees. First we prove soundness and completeness for non-recursive types. Soundness is then derived by observing that a successful execution of the algorithm on some input must also be successful on all the finite approximations of the input. Completeness is proven by examining a failing execution tree, and concluding that the trees corresponding to the input must have been different to start with.

3.3.1 Lemma *(Derived structural computational rules)*
Given the assumptions in 3.1.2, 3.2 and 3.2.3 we have:

Σ-weaken
If $\vdash_A \Sigma,\varepsilon \supset t<s$ and $\Sigma \cup \Sigma' \# \varepsilon$ then $\vdash_A \Sigma \cup \Sigma',\varepsilon \supset t<s$.

Σ-strengthen
If $\vdash_A \Sigma \cup \Sigma',\varepsilon \supset t<s$ and $Reach(t \rightarrow s,\varepsilon) \cap Vars(\Sigma') = \emptyset$ then $\vdash_A \Sigma,\varepsilon \supset t<s$.

ε-weaken
If $\vdash_A \Sigma,\varepsilon \supset t<s$, $Reach(t \rightarrow s,\varepsilon) \cap Dom(\varepsilon') = \emptyset$, $\Sigma \# \varepsilon \cup \varepsilon'$, and $Dom(\varepsilon) \cap Dom(\varepsilon') = \emptyset$ then $\vdash_A \Sigma,\varepsilon \cup \varepsilon' \supset t<s$.

ε-strengthen
If $\vdash_A \Sigma,\varepsilon \cup \varepsilon' \supset t<s$ and $Reach(t \rightarrow s,\varepsilon) \cap Dom(\varepsilon') = \emptyset$ then $\vdash_A \Sigma,\varepsilon \supset t<s$.

3.3.2 Proposition *(Completeness of $<_A$ for non-rec. types)*
Given $\alpha,\beta \in Tp$ non-recursive types then
$\alpha <_T \beta \Rightarrow \alpha <_A \beta$.

Proof (sketch)
Let $\varepsilon \triangleq E\alpha \cup E\beta$. We show $\alpha <_T \beta \Rightarrow \forall \Sigma.\ \Sigma \# \varepsilon \Rightarrow \vdash_A \Sigma,\varepsilon \supset \alpha^* < \beta^*$ by induction on the structure of α and β. □

3.3.3 Proposition *(Soundness of $<_A$ for non-rec. types)*
Given $\alpha,\beta \in Tp$ non-recursive types then:
$\alpha <_A \beta \Rightarrow \alpha <_T \beta$.

Proof (sketch)
We show $\vdash_A \emptyset,\varepsilon \supset \alpha^* < \beta^* \Rightarrow \alpha <_T \beta$, where $\varepsilon \triangleq E\alpha \cup E\beta$, by induction on the structure of α and β. □

3.3.4 Lemma *(Uniformity of $<_A$)*
Let $\alpha,\beta \in Type$. If $\alpha <_A \beta$ then $\forall k.\alpha_{|k} <_A \beta_{|k}$.

Proof (sketch)
Given any k, from the execution tree of $\alpha <_A \beta$ it is possible to extract a successful execution tree for $\alpha_{|k} <_A \beta_{|k}$. The point is that the use of the $(assmp_A)$ rule can be arbitrarily delayed by repeating a certain pattern of computation. □

3.3.5 Proposition *(Soundness of $<_A$)*
Let $\alpha,\beta \in$ Type; if $\alpha <_A \beta$ then $\alpha <_T \beta$.

Proof
From 3.3.4 we have: $\alpha <_A \beta \Rightarrow \forall k.\alpha_{|k} <_A \beta_{|k}$. From 3.3.3 and the definition of $<_T$ we have: $\forall k.\alpha_{|k} <_{fin} \beta_{|k}$ and $\alpha <_T \beta$. □

3.3.6 Lemma *(Faithfulness of $<_A$ w.r.t. paths)*
Let $lead(\alpha,\varepsilon) \triangleq Sol(\alpha,\varepsilon)(nil)$ be the first label of α in ε (that is, skipping initial variables in α,ε).

Let $\Sigma,\varepsilon \supset \alpha <\beta$ be the root of an execution tree, terminating with success or failure leaves, obtained from the rules in 3.2. Every node $\Sigma',\varepsilon \supset \alpha' <\beta'$ in the execution tree determines a path π from the root to itself, given by considering the occurrences of $(\rightarrow_A)$ and ignoring the other rules. Then:

1) Either α' and β' are both (bound) type variables, or neither is.

2) $T\alpha(\pi) = lead(\alpha',\varepsilon)$ and $T\beta(\pi) = lead(\beta',\varepsilon)$.

Proof
By induction on the depth of the execution tree. □

3.3.7 Proposition *(Completeness of $<_A$)*
Let $\alpha,\beta \in$ Type; if $\alpha <_T \beta$ then $\alpha <_A \beta$.

Proof
We show $\neg\, \alpha <_A \beta \Rightarrow \neg\, T\alpha <_\infty T\beta$.

By assumption, we have an execution tree for $\alpha <_A \beta$ which contains a failure node $\Sigma,\varepsilon \supset \alpha' <\beta'$, determining a path π as in Lemma 3.3.6. By 3.3.6.(2), $T\alpha(\pi) = lead(\alpha',\varepsilon)$ and $T\beta(\pi) = lead(\beta',\varepsilon)$. Hence we have a *common* path in $T\alpha$ and $T\beta$ corresponding to the failure node. The following table summarizes the possible cases for α',β' where the entry indicates either failure or the rule being applied by the algorithm; the n.a. (not applicable) cases come from 3.3.6.(1).

$\alpha' \backslash \beta'$	$\bot$	$\top$	s	b	$\beta' \rightarrow \beta''$
$\bot$	$\bot$	$\bot$	n.a.	$\bot$	$\bot$
$\top$	fail	$\top$	n.a.	fail	fail
t	n.a.	n.a.	assmp-μ	n.a.	n.a.
a	fail	$\top$	n.a.	var-fail	fail
$\alpha' \rightarrow \alpha''$	fail	$\top$	n.a.	fail	$\rightarrow$

Every "fail" in the algorithm corresponds to a situation where the two trees cannot be in the inclusion relation. □

4. Typing Rules

In this section we introduce a certain number of axioms and rules that induce an equivalence and a preorder on the collection of types. Moreover, we discuss the relationship with the corresponding model, tree, and algorithm notions.

4.1 Type Equivalence Rules

We say that a type α is *contractive* in the type variable t if either t does not occur free in α, or α can be rewritten via unfolding as a type of the shape $\alpha_1 \rightarrow \alpha_2$. We write this fact as $\alpha\downarrow t$.

It is now easy to observe that the contractiveness of α in t is a sufficient (and necessary) condition to enforce the contractiveness of the following functional on the space Tree(L) (2.3):

$$G_{\alpha,t}(A) \triangleq [A/t]T\alpha \qquad A \in Tree(L)$$

($[A/t]T\alpha$ denotes the substitution of the tree A for the occurrences of t in $T\alpha$.)

This remark suggests the following rule that is generalized to a larger calculus in [Cardelli Longo 90]:

$$\text{(contract)} \qquad [\beta/t]\alpha = \beta,\ [\beta'/t]\alpha = \beta',\ \alpha\downarrow t \ \Rightarrow\ \beta = \beta'$$

In words, if two types β and β' are fixpoints of the same functional $\alpha[t]$, then they are equal since contractive functionals have unique fixpoints. This rule was also inspired by a standard proof technique for bisimulation [Park 81].

Finally it is convenient to identify $\mu t.t = \bot$.

Hence, in this section we consider the equivalence:

$$\vdash \alpha = \beta \qquad (\text{or } \alpha =_R \beta)$$

meaning that $\alpha = \beta$ can be derived in the congruence induced by the (contract) rule, and the (fold-unfold) and (μ-$\bot$) axioms.

4.1.1 Proposition *(Soundness of the eq. rules w.r.t. trees)*
$\alpha =_R \beta \ \Rightarrow\ T\alpha = T\beta$

Proof
Immediate by the previous considerations. □

4.1.2 Derived Rules

By means of (contract) and (fold-unfold) it is possible to prove new interesting equivalences, for example:

(1) $\mu t.s \rightarrow t = \mu t.s \rightarrow (s \rightarrow t)$
(2) $\mu t.\mu s.\alpha = \mu v.[v/t,v/s]\alpha$ (μ-contraction)

4.1.3 Reduction to Canonical Form

It easy to show that any recursive type is provably equivalent ($=_R$) to a type in canonical form. The strategy can be described as follows:

- Use unfold to remove μ's that do not bind variables.
- Use μ-contraction to reduce sequences of μ's to one μ.
- Use μ-$\bot$ to reduce to $\bot$ all subtypes of the shape $\mu t.t$.

4.2 Completeness of Equivalence Rules

By the strong connection between regular trees and recursive types we show that any time two recursive types α, β have the same tree expansion $T\alpha = T\beta$, then we can conclude $\vdash \alpha = \beta$.

First we show how to solve systems of type equations. Then we introduce the notion of *equational characterization* of a type; that is, how to characterize a type by a system of type equations. Finally we use equational characterizations to prove the completeness theorem.

In this section we use the following notation. If γ has free variables $\{u_1..u_p\} \subseteq \{t_1..t_n\}$, then we write $\gamma[\alpha_1...\alpha_n]$ for the substitution $[\alpha_1/t_1 \ ...\ \alpha_n/t_n]\gamma$. In particular, $\gamma[t_1...t_n]$ emphasizes a superset of the free variables of γ.

4.2.1 Lemma *(A system of equations has a solution, by iterated elimination)*

Every system of n equations in n variables:

$t_i=\gamma_i[t_1...t_n]$ $(i\in 1..n)$

has a solution in the congruence induced by the axiom (fold-unfold). That is, there are $\alpha_1...\alpha_n$ such that $\vdash \alpha_i=\gamma_i[\alpha_1...\alpha_n]$ $(i\in 1..n)$.

4.2.2 Lemma *(A system of contractive equations has a unique solution)*

Assume that, for $i\in 1..n$, we have two sets of types α_i, β_i, related by two systems of equations:

$\vdash \alpha_i=\gamma_i[\alpha_1...\alpha_n]$ $\vdash \beta_i=\gamma_i[\beta_1...\beta_n]$

such that $\gamma_i[t_1...t_n]\downarrow tj$ for $i,j\in 1..n$. Then, for all i: $\vdash \alpha_i=\beta_i$.

4.2.3 Definition

A *node context* $p[t_1...t_n]$ for $p\in L$ (see 2.3) is a type of the form $p(u_1..u_{\#p})$, where #p is the arity of p, and $\{u_1..u_{\#p}\}\subseteq\{t_1...t_n\}$. (Hence, a node context is contractive in each t_i.)

4.2.4 Definition

A type $\alpha\in$Type is *equationally characterized (eq. char.)* if there are types $\alpha_1..\alpha_n$ with $\alpha\equiv\alpha_1$, and there are node contexts $p_i[t_1...t_n]$, $i\in 1..n$, for which $\vdash \alpha_i=p_i[\alpha_1...\alpha_n]$.

An equation $p_j[t_1...t_n]$ in a system is reachable from a variable t_k if k=j, or if it is reachable from the variables in $p_k[t_1...t_n]$ (see 3.1.2). An equation is reachable from another if it is reachable from any of the variables in the other.

4.2.5 Lemma *(Building an equational characterization)*

Every term $\alpha\in$Type has an equational characterization such that all equations are reachable from the first one.

Proof (sketch)

The construction is basically the same as the one in 3.1.4. It is enough to prove that every term $\gamma\in\mu$Tp is equationally characterized by induction on the structure of γ. Then the lemma follows by 4.1.3, and by the invariance of equational characterization modulo provable equivalence. □

4.2.6 Lemma

Assume $T\alpha=T\beta$ and $\vdash \alpha=p(\alpha_1..\alpha_{\#p})$, $\vdash \beta=q(\beta_1..\beta_{\#q})$, where $p,q\in L$. Then p=q and $T\alpha_i=T\beta_i$ for all $i\in 1..\#p$.

Proof

By soundness, $T\alpha=Tp(\alpha_1..\alpha_{\#p})$ and $T\beta=Tq(\beta_1..\beta_{\#q})$. Hence, p=q and $T\alpha_i=T\beta_i$ by definition of T. □

4.2.7 Theorem *(Completeness of type equivalence rules)*

If $T\alpha=T\beta$ then $\alpha=_R\beta$

Proof

The idea of the proof is as follows: given α and β such that $T\alpha=T\beta$ we produce their corresponding equational characterizations, say $ec(\alpha)$ and $ec(\beta)$. By a collapse of "equivalent" equations we derive a new equational characterization $ec(\gamma)$. The solutions of the (smaller) system associated with $ec(\gamma)$ can be replicated to produce solutions for the systems associated with $ec(\alpha)$ and $ec(\beta)$. Hence we can apply twice Lemma 4.2.2 (uniqueness of solutions) and then transitivity to conclude $\alpha=_R\beta$.

Let $T\alpha=T\beta$; by Lemma 4.2.5, α,β are equationally characterized by α_i, $t_i=p_i[t_1...t_n]$ and β_j, $t_j=q_j[t_1...t_m]$ so that all equations are reachable from the first ones.

From these α_i,β_j we generate a sequence of pairs (A^h,B^h) where A^h,B^h are equivalence classes of α_i and β_j respectively. Moreover, for each h, $\alpha_{i1},\alpha_{i2}\in A^h$, and $\beta_{j1},\beta_{j2}\in B^h$, we shall have the invariant $T\alpha_{i1}=T\alpha_{i2}=T\beta_{j1}=T\beta_{j2}$.

We start with the pair $(\{\alpha\},\{\beta\})$. At each step we consider all the pairs α_i,β_j such that $\alpha_i\in A^h$ and $\beta_j\in B^h$ for some h. We indicate by $\alpha_{(i',i'')}$ some α_i depending on both i' and i"; similarly for $\beta_{(j',j'')}$. If $\alpha_i = p_i(\alpha_{(i,1)}...\alpha_{(i,\#p_i)})$ and $\beta_j = q_j(\beta_{(j,1)}...\beta_{(j,\#q_j)})$, we have, by Lemma 4.2.6, $p_i=q_j$ and $T\alpha_{(i,1)}=T\beta_{(j,1)}$ ... $T\alpha_{(i,\#p_i)}=T\beta_{(j,\#p_i)}$. We add all the pairs $\alpha',\beta'\in\{\alpha_{(i,1)},\beta_{(j,1)}$ ... $\alpha_{(i,\#p_i)},\beta_{(j,\#p_i)}\}$ in the following way, respecting the invariant above:

- if $\alpha'\in A^h$ and $\beta'\in B^h$ for some h, then nothing is done;
- else, if $\alpha'\in A^{h1}$, and $\beta'\in B^{h2}$, with $h1\neq h2$, then we replace the pairs (A^{h1},B^{h1}) and (A^{h2},B^{h2}) by $(A^{h1}\cup A^{h2},B^{h1}\cup B^{h2})$;
- else, if $\alpha'\in A^h$ we replace the pair (A^h,B^h) by $(A^h,B^h\cup\{\beta'\})$;
- else, if $\beta'\in B^h$ we replace the pair (A^h,B^h) by $(A^h\cup\{\alpha'\},B^h)$;
- else we add a new pair $(\{\alpha'\},\{\beta'\})$.

We stop when the list of pairs no longer changes. This process terminates because there are at most n·m pairs to consider.

The process above produces two partitions of α_i and β_j of size $k\leq n$, $k\leq m$, for some k. These are total partitions since all equations are reachable from the first ones. These partitions determine two functions $\sigma:1..n\to 1..k$ and $\pi:1..m\to 1..k$ such that:

- $\sigma(i)=\pi(j)\Leftrightarrow \alpha_i\in A^h$ $\beta_i\in B^h$ for some h
- $\sigma(i1)=\sigma(i2)\Leftrightarrow \alpha_{i1},\alpha_{i2}\in A^h$ for some h
- $\pi(i1)=\pi(i2)\Leftrightarrow \beta_{i1},\beta_{i2}\in B^h$ for some h

Given these partitions, we now define a system of k equations $t_h = r_h[t_1..t_k]$, which will turn out to be equivalent both to the p_i and the q_j systems. For $h\in 1..k$ we have:

$t_h = r_h[t_1..t_k]$ where
$r_{\sigma(i)}[t_1..t_k]\equiv p_i[t_{\sigma(1)}...t_{\sigma(n)}]$
$r_{\pi(j)}[t_1..t_k]\equiv q_j[t_{\pi(1)}...t_{\pi(m)}]$

We need to argue that this is a proper definition, since we can have, for example, $\sigma(i)=\pi(j)$ for some i,j. We show that when this happens, we also have by construction that $r_{\sigma(i)}[t_1..t_k]\equiv r_{\pi(j)}[t_1..t_k]$. Similarly for the other possible conflicts: $\sigma(i1)=\sigma(i2)$ for some i1,i2, and $\pi(i1)=\pi(i2)$ for some j1,j2. To show these facts, we further investigate the properties of σ and π.

- $\sigma(1) = \pi(1)$ since α,β start in the same pair (A^1,B^1).

- if $\sigma(i) = \pi(j)$ then $p_i \equiv q_j$. Moreover, let $\alpha_i = p_i[\alpha_1 ... \alpha_n] \equiv p_i(\alpha_{(i,1)} ... \alpha_{(i,\#p_i)})$ and $\beta_j = q_j[\beta_1 ... \beta_m] \equiv q_j(\beta_{(j,1)} ... \beta_{(j,\#q_j)})$ be the i-th and j-th equations in the respective systems. Then $\alpha_i \in A^h$, $\beta_j \in B^h$ for some h (property above); the pair α_i,β_j was considered in the process above; that is, the pairs $\alpha_{(i,1)},\beta_{(j,1)}$ ... $\alpha_{(i,\#p_i)},\beta_{(j,\#p_i)}$ were also added to the list. Therefore $\sigma(i,1)=\pi(j,1)$ ... $\sigma(i,\#p_i)=\pi(j,\#q_j)$, and $p_i(t_{\sigma(i,1)} ... t_{\sigma(i,\#p_i)}) \equiv q_j(t_{\pi(j,1)} ... t_{\pi(j,\#q_j)})$. This is the same as saying $p_i[t_{\sigma(1)} ... t_{\sigma(n)}] \equiv q_j[t_{\pi(1)} ... t_{\pi(m)}]$.

- if $\sigma(i1) = \sigma(i2)$ then $p_{i1} \equiv p_{i2}$. Moreover, let $\alpha_{i1} = p_{i1}(\alpha_{(i1,1)} ... \alpha_{(i1,\#p_{i1})})$ and $\alpha_{i2} = p_{i2}(\alpha_{(i2,1)} ... \alpha_{(i2,\#p_{i2})})$ be the i1-th and i2-th equations in the α system. Then $\alpha_{i1},\alpha_{i2} \in A^h$ for some h (property above). Consider any $\beta_j \in B^h$; the pairs α_{i1},β_j , α_{i2},β_j were considered in the process above, that is the pairs $\alpha_{(i1,1)},\beta_{(j,1)}$, $\alpha_{(i2,1)},\beta_{(j,1)}$ were also added to the list. Therefore $\sigma(i1,1)=\pi(j,1)=\sigma(i2,1)$, and similarly up to $\sigma(i1,\#p_{i1})=\sigma(i2,\#p_{i2})$. Hence: $p_{i1}(t_{\sigma(i1,1)} ... t_{\sigma(i1,\#p_{i1})}) \equiv p_{i2}(t_{\sigma(i2,1)} ... t_{\sigma(i2,\#p_{i2})})$. This is the same as saying $p_{i1}[t_{\sigma(1)} ... t_{\sigma(n)}] \equiv p_{i2}[t_{\sigma(1)} ... t_{\sigma(n)}]$.

- similarly for $\pi(i1) = \pi(i2)$.

Hence we conclude:

if $\sigma(i) = \pi(j)$ then
$r_{\sigma(j)}[t_1..t_k] \equiv p_i[t_{\sigma(1)} ... t_{\sigma(n)}] \equiv q_j[t_{\pi(1)} ... t_{\pi(m)}] \equiv r_{\pi(j)}[t_1..t_k]$
if $\sigma(i1)=\sigma(i2)$ then
$r_{\sigma(i1)}[t_1..t_k] \equiv p_{i1}[t_{\sigma(1)} ... t_{\sigma(n)}] \equiv p_{i2}[t_{\sigma(1)} ... t_{\sigma(n)}] \equiv r_{\sigma(i2)}[t_1..t_k]$
similarly for $\pi(j1)=\pi(j2)$

Now by Lemma 4.2.1 we can construct a solution of the system $t_h = r_h[t_1..t_k]$; that is, we can obtain $\gamma_1..\gamma_k$ such that $\vdash \gamma_h = r_h[\gamma_1..\gamma_k]$.

Then $\vdash \gamma_{\sigma(i)} = r_{\sigma(i)}[\gamma_1..\gamma_k] \equiv p_i[\gamma_{\sigma(1)} ... \gamma_{\sigma(n)}]$ for all i. Therefore, the γ's (when appropriately replicated) satisfy the same system as the α's, and by Lemma 4.2.2 we have $\vdash \alpha_i = \gamma_{\sigma(i)}$. Similarly, the γ's satisfy the β's system, and $\vdash \beta_j = \gamma_{\pi(j)}$. Moreover, $\sigma(1) = \pi(1)$, hence $\vdash \alpha \equiv \alpha_1 = \gamma_{\sigma(1)} = \gamma_{\pi(1)} = \beta_1 \equiv \beta$ by transitivity. □

This constructive proof is based on the one in [Salomaa 66] (see also [Milner 84]), but differs in an important point as, in addition, we must deal with equivalence classes of types.

4.3 Subtyping Rules

At first it is not clear how to define a rule for the subtyping of recursive types that is sufficiently powerful. In particular, observe that the computational rule (μ_A) in section 3.2 does not have any apparent logical meaning as the assumption is always valid under a classical reading of the entailment relation.

We now introduce a rule, (μ_R), whose soundness is clear. Later, in section 4.4, we will show that in conjunction with the type equivalence rules, (μ_R) leads to a subtyping system complete with respect to the tree ordering.

We denote with Γ a set $\{t_1<s_1, ..., t_n<s_n\}$ of subtyping assumptions on type variables. We write a subtype judgment as: $\Gamma \supset \alpha<\beta$.

Define a formal system for deriving this kind of judgments as follows; this is based on the $\alpha=\beta$ congruence in 4.1:

(eq_R) $\alpha = \beta \Rightarrow \Gamma \supset \alpha<\beta$
$(trans_R)$ $\Gamma \supset \alpha<\beta$, $\Gamma \supset \beta<\gamma \Rightarrow \Gamma \supset \alpha<\gamma$
$(assmp_R)$ $t<s \in \Gamma \Rightarrow \Gamma \supset t<s$
$(\bot_R)$ $\Gamma \supset \bot < \alpha$
$(\top_R)$ $\Gamma \supset \alpha < \top$
$(\rightarrow_R)$ $\Gamma \supset \alpha'<\alpha$, $\Gamma \supset \beta<\beta' \Rightarrow \Gamma \supset \alpha\rightarrow\beta < \alpha'\rightarrow\beta'$
(μ_R) $\Gamma \cup \{t<s\} \supset \alpha<\beta \Rightarrow \Gamma \supset \mu t.\alpha < \mu s.\beta$
with t only in α; s only in β; t,s not in Γ

We say $\alpha <_R \beta$ if we can derive $\varnothing \supset \alpha<\beta$. The last rule was proposed in [Cardelli 86] in the specification of the Amber programming language as a first attempt to define a theory for the subtyping of recursive types.

4.3.1 Proposition *(Soundness of the rule ordering w.r.t. the tree ordering)*
If $\alpha <_R \beta$ then $\alpha <_T \beta$.

Proof

We prove the more general statement:

If $\vdash_R \{t_1<s_1, .., t_n<s_n\} \supset \alpha<\beta$ and $\alpha_1 <_T \beta_1, .., \alpha_n <_T \beta_n$
so that $\{t_1,s_1, .., t_n,s_n\} \cap FV(\alpha_1,\beta_1, .., \alpha_n,\beta_n) = \varnothing$
then $[\alpha_1/t_1, \beta_1/s_1, .., \alpha_n/t_n, \beta_n/s_n]\alpha$
$<_T [\alpha_1/t_1, \beta_1/s_1, .., \alpha_n/t_n, \beta_n/s_n]\beta$.

The proof goes by induction on the length of the derivation $\vdash_R$. The only interesting cases arise for (μ_R) and (eq_R).

For brevity we write lists such as $t_1<s_1, ..., t_n<s_n$ in the form $t_i<s_i$ for a free i.

Case (μ_R) $\{t_i<s_i, t<s\} \supset \alpha<\beta \Rightarrow \{t_i<s_i\} \supset \mu t.\alpha < \mu s.\beta$
with $t \notin FV(\beta)$; $s \notin FV(\alpha)$; $t,s \neq t_i,s_i$ for any i.
By induction hypothesis:
$\forall \alpha_i <_T \beta_i, \underline{\alpha} <_T \underline{\beta}$ s.t. $\{t_i,s_i,t,s\} \cap FV(\alpha_i,\beta_i,\underline{\alpha},\underline{\beta}) = \varnothing$.
$[\alpha_i/t_i, \beta_i/s_i, \underline{\alpha}/t]\alpha <_T [\alpha_i/t_i, \beta_i/s_i, \underline{\beta}/s]\beta$
Define $\alpha^0 \triangleq \bot$ $\alpha^{n+1} \triangleq [\alpha_i/t_i, \beta_i/s_i, \alpha^n/t]\alpha$
$\beta^0 \triangleq \bot$ $\beta^{n+1} \triangleq [\alpha_i/t_i, \beta_i/s_i, \beta^n/s]\beta$
Applying the induction hypothesis with $\underline{\alpha}=\alpha^n$, $\underline{\beta}=\beta^n$ we obtain
$\alpha^{n+1} <_T \beta^{n+1}$ for every n.
For every k we can then choose an n sufficiently large so that:
$(\mu t.[\alpha_i/t_i, \beta_i/s_i]\alpha)_{|k} =_T \alpha^n{}_{|k} <_T \beta^n{}_{|k}$
$=_T (\mu s.[\alpha_i/t_i, \beta_i/s_i]\beta)_{|k}$
Hence, by definition of $<_T$ for recursive types, we have shown:
$[\alpha_i/t_i, \beta_i/s_i](\mu t.\alpha) <_T [\alpha_i/t_i, \beta_i/s_i](\mu s.\beta)$

Case (eq_R) $\alpha =_R \beta \Rightarrow \{t_i<s_i\} \supset \alpha<\beta$
Since $=_R$ is a congruence, we have
$[\alpha_i/t_i, \beta_i/s_i]\alpha =_R [\alpha_i/t_i, \beta_i/s_i]\beta$.
By soundness of $=_R$ we have
$[\alpha_i/t_i, \beta_i/s_i]\alpha =_T [\alpha_i/t_i, \beta_i/s_i]\beta$.
Finally,
$[\alpha_i/t_i, \beta_i/s_i]\alpha <_T [\alpha_i/t_i, \beta_i/s_i]\beta$
since $<_T$ is a preorder. □

4.4 Completeness of Subtyping Rules.

In proving the completeness of the subtyping rules w.r.t. the tree ordering, it seems helpful to go through the algorithm. The rather obvious approach of extracting a proof from a successful execution tree is complicated by the lack of correspondence between the computational rule (μ_A) and the rule (μ_R), as the former can be applied repeatedly on the same variable, whereas the latter can be applied at most once.

So, while in the proof of completeness of the type equivalence rules we relied on minimization techniques, here we introduce some redundancy in order to find equivalent systems that never expand twice the same variable by means of (μ_A).

If $\vdash_A \Sigma,\varepsilon\supset t<s$ (see 3.2.3), we say that (the successful execution tree of) the initial goal $\Sigma,\varepsilon\supset t<s$ has the *one-expansion property* iff the following is true: for every $t\in Dom(\varepsilon)$ and for each path p of the execution tree, t is expanded in a (μ_A) node of p at most once.

It follows that with one-expansion, each variable can be inserted in Σ in a unique way, so that for each pair of assumptions $t_1<s_1$, $t_2<s_2 \in \Sigma$ we have that t_1,s_1,t_2,s_2 are pairwise distinct. Moreover, if we consider two (μ_A) nodes $\Sigma,\varepsilon\supset t_1<s_1$, $\Sigma,\varepsilon\supset t_2<s_2$ on the same path then t_1,s_1,t_2,s_2 are pairwise distinct, and if we consider a (μ_A) node $\Sigma,\varepsilon\supset t_1<s_1$ and an ($assmp_A$) node $\Sigma,\varepsilon\supset t_2<s_2$ on the same path then either $t_1\equiv t_2$, $s_1\equiv s_2$ or t_1,s_1,t_2,s_2 are pairwise distinct.

4.4.1 Lemma *(Putting recursions in lockstep)*

If $\vdash_A \Sigma,\varepsilon\supset t<s$ then there are θ, r, u such that $\vdash_A \Sigma,\theta\supset r<u$, $Sol(r,\theta) = Sol(t,\varepsilon)$, $Sol(u,\theta) = Sol(s,\varepsilon)$ and $\Sigma,\theta\supset r<u$ satisfies the one-expansion property.

Proof

Given the initial goal $\Sigma,\varepsilon\supset t<s$ and the related successful execution tree we build a new judgment $\Sigma,\theta\supset r<u$ such that the following properties hold:

(a) $\Sigma,\theta\supset r<u$ is an initial goal.
(b) $Sol(r,\theta) = Sol(t,\varepsilon)$ and $Sol(u,\theta) = Sol(s,\varepsilon)$.
(c) $\vdash_A \Sigma,\theta\supset r<u$, and the execution tree is equal to the one for $\Sigma,\varepsilon\supset t<s$ modulo variable renaming.
(d) $\Sigma,\theta\supset r<u$ satisfies the one-expansion property.

First we build the execution tree of $\Sigma,\varepsilon\supset t<s$. Then we associate with every node of the tree a couple (r, u) (or (u, r) on negative branches) of fresh variables with the following constraint; with every assumption leaf for t<s we associate the same pair of variables as with the μ node where the assumption t<s has been introduced into Σ (if any).

Next generate θ according to the following cases:

Case (μ-$\bot$). Say we are in the situation: $\Sigma',\varepsilon\supset\bot<\beta \Rightarrow \Sigma',\varepsilon\supset t<s_0$ where $\varepsilon(t)=\bot$. If (r,u_0) is the pair of variables associated with the μ-node add the equations:

$r=\bot$

$[u_0/s_0,u_1/s_1, ..., u_n/s_n](s_i=\varepsilon(s_i))$ for $i\in 0..n$

where $u_1...u_n$ are fresh variables and $s_1...s_n$ are the variables reachable from s_0 in the system ε, that is $\{s_1...s_n\} = Reach(s_0, \varepsilon)\cap Dom(\varepsilon)$.

Case (μ-$\top$). Analogous.

Case (μ-var). Say we are in the situation $\Sigma',\varepsilon\supset a<a \Rightarrow \Sigma',\varepsilon\supset t<s$. If (r,u) is the pair of variables associated with the μ-node, we add a pair of equations: r=a, u=a.

Case (μ-$\rightarrow$). Say we are in the situation:

$\Sigma'\cup\{t<s\},\varepsilon\supset s_1<t_1$, $\Sigma'\cup\{t<s\},\varepsilon\supset t_2<s_2$

$\Rightarrow \Sigma'\cup\{t<s\},\varepsilon\supset t_1\rightarrow t_2<s_1\rightarrow s_2 \Rightarrow \Sigma',\varepsilon\supset t<s$

where we have the fresh variables r,r_1,r_2 for t,t_1,t_2 and u,u_1,u_2 for s,s_1,s_2 (the variables associated to an $\rightarrow$-node are inessential) then we generate the equations $r=r_1\rightarrow r_2$ and $u=u_1\rightarrow u_2$.

Case (μ-assmp1). Say we are in the situation: $\Sigma',\varepsilon\supset a<b \Rightarrow \Sigma',\varepsilon\supset t<s$ where $a<b\in\Sigma$. If (r,u) is the pair of variables associated with the μ-node, we add a pair of equations: r=a, u=b.

Case (μ-assmp2). Finally, if we visit a node in which we apply the rule ($assmp_A$) w.r.t. an assumption added during the computation then we do not generate any equation. In fact, the equations corresponding to those variables are defined in the corresponding μ-node in which the assumption was made.

Let us now consider the properties (a-d):

(a) Follows from the use of fresh variables.

(b) In the first place one establishes a relation R, say, between the variables reachable from t and those reachable from r. In general we will have a situation in which a variable t may correspond to many variables $r_1...r_n$. Next, prove by induction on the lowest level of the appearance of r in the execution tree and $|\pi|$ that $(t, r)\in R$ implies $Sol(r,\theta)(\pi) = Sol(t,\varepsilon)(\pi)$.

(c) By construction at each step we can apply the same computational rule.

(d) This is a consequence of the constraint on the assignment of fresh variables to nodes. □

4.4.2 Lemma *(From the execution tree to the proof tree)*

If $\vdash_A \Sigma,\varepsilon\supset t<s$ (see 3.2.3) and its execution tree has the one-expansion property, then $\vdash_R \Sigma\supset\langle t,\varepsilon\rangle < \langle s,\varepsilon\rangle$.

Proof

We proceed by induction on the depth k of the successful execution tree of an initial goal $\Sigma,\varepsilon\supset t<s$. Depth is measured by the number of adjacent pairs of nodes (μ_A)-($\rightarrow_A$) in the longest branch from the root. In the inductive case, each subgoal is converted into an initial goal of the same depth, in order to apply the induction hypothesis.

Case k=0.

The tree consists of a (μ_A) root (since the goal is initial) and a single leaf which is either ($assmp_A$), ($\bot_A$), ($\top_A$), or (var_A). Then after the application of the (μ_A) rule, with $s,t\in Dom(\varepsilon)$, we are in a terminal case $\Sigma\cup\{t<s\},\varepsilon\supset\varepsilon(t)<\varepsilon(s)$.

Subcase ($assmp_A$). $\Sigma\cup\{t<s\},\varepsilon\supset a<b$, where $\varepsilon(t)=a$, $\varepsilon(s)=b$, and $a<b\in\Sigma$. Then $a,b\notin Dom(\varepsilon)$ (by definition of Tenv), and $\langle t,\varepsilon\rangle=\mu t.a=a$, $\langle s,\varepsilon\rangle=\mu s.b=b$. By ($assmp_R$),

$a<b \in \Sigma \Rightarrow \vdash_R \Sigma \supset a<b$. Conclude by (eq_R): $\vdash_R \Sigma \supset \mu t.a<a$, $\vdash_R \Sigma \supset b<\mu s.b$, and $(trans_R)$.

Subcase $(\bot_A)$. $\Sigma \cup \{t<s\}, \varepsilon \supset \bot<\varepsilon(s)$, where $\varepsilon(t)=\bot$. Then $\langle t,\varepsilon\rangle=\mu t.\bot=\bot$ and we have $\vdash_R \Sigma \supset \bot < \langle s,\varepsilon\rangle$. Conclude by (eq_R) and $(trans_R)$.

Subcase $(\top_A)$. Similar.

Subcase (var_A). $\Sigma \cup \{t<s\}, \varepsilon \supset a<a$, where $\varepsilon(t)=\varepsilon(s)=a$ and $a \notin Dom(\varepsilon)$. Then $\langle t,\varepsilon\rangle=\mu t.a=a=\mu s.a=\langle s,\varepsilon\rangle$. We can apply (eq_R): $\Sigma \supset a<a$, then conclude with (eq_R) and $(trans_R)$.

Case k>0.

The tree has a (μ_A) root with a $(\rightarrow_A)$ child, hence $\varepsilon(t)=t_1 \rightarrow t_2$, $\varepsilon(s)=s_1 \rightarrow s_2$, where by definition of Tenv $t_1,t_2,s_1,s_2 \in Dom(\varepsilon)$:

$\Sigma \cup \{t<s\}, \varepsilon \supset s_1<t_1$, $\Sigma \cup \{t<s\}, \varepsilon \supset t_2<s_2$
$\Rightarrow \Sigma \cup \{t<s\}, \varepsilon \supset t_1 \rightarrow t_2 < s_1 \rightarrow s_2$
$\Rightarrow \Sigma, \varepsilon \supset t<s$

We initially focus on one of the subgoals of depth k-1:

(A) $\Sigma \cup \{t<s\}, \varepsilon \supset t_2<s_2$

Let us consider the following goal (B), which we intend to subject, instead of (A), to the induction hypothesis:

(B) $\Sigma \cup \{t<s\}$, $\varepsilon' \supset \sigma(t_2)<\sigma(s_2)$

where $\sigma \triangleq [t'/t, s'/s]$ is a substitution with fresh variables t' and s', and $\varepsilon' \triangleq \sigma(\varepsilon \backslash t \backslash s) \cup \{t'=t, s'=s\}$.

First we show that the goal (B) is initial. Since $\vdash_A \Sigma, \varepsilon \supset t<s$ is initial we have:

$Vars(\Sigma) \cap Dom(\varepsilon)=\varnothing$
$\varepsilon=\varepsilon_1 \cup \varepsilon_2$ with $Dom(\varepsilon_1) \cap Dom(\varepsilon_2)=\varnothing$,
such that $t \in Dom(\varepsilon_1)$, $s \in Dom(\varepsilon_2)$

Hence we also have:

$t_1,t_2 \in Dom(\varepsilon_1)$ (only); $s_1,s_2 \in Dom(\varepsilon_2)$ (only)
$\varepsilon'=\varepsilon'_1 \cup \varepsilon'_2$
where $\varepsilon'_1 \triangleq \sigma(\varepsilon_1 \backslash t) \cup \{t'=t\}$, $\varepsilon'_2 \triangleq \sigma(\varepsilon_2 \backslash s) \cup \{s'=s\}$

From which we conclude:

$Vars(\Sigma \cup \{t<s\}) \cap Dom(\varepsilon')=\varnothing$
$Dom(\varepsilon'_1) \cap Dom(\varepsilon'_2)=\varnothing$
$\sigma(t_2) \in Dom(\varepsilon'_1)$, because:
if $t_2 \equiv t$ then $\sigma(t_2) \equiv t'$ and $t' \in Dom(\varepsilon'_1)$;
($t_2 \equiv s$ is not possible)
if $t_2 \not\equiv t$ then $\sigma(t_2) \equiv t_2$; since $t_2 \in Dom(\varepsilon_1)$,
we have $\sigma(t_2) \in Dom(\varepsilon'_1)$
$\sigma(s_2) \in Dom(\varepsilon'_2)$, similarly.

Second, let Tree(A) be the execution subtree of root (A), and Tree(B) be the execution tree of root (B). We show, by induction on the *length* of the longest path in Tree(A), that we can build a tree T such that: (1) T has the same *depth* as Tree(A); (2) T succeeds; (3) T expands the same variables as Tree(A) in (μ_A) nodes, with the exception of t',s'; (4) T has the one-expansion property; and (5) T = Tree(B). (Hence, we also have $\vdash_A$ (B).)

We proceed by induction on each subgoal $\underline{A} = \underline{\Sigma} \cup \{t<s\}, \varepsilon \supset \underline{\alpha}<\underline{\beta}$ of Tree(A), for which we build a subtree $\underline{T}$ of the shape $\underline{\Sigma} \cup \{t<s\}, \varepsilon' \supset \sigma(\underline{\alpha})<\sigma(\underline{\beta})$.

For the case $(assmp_A)$, by the properties of one-expansion we only have to consider the cases when either $\underline{t} \equiv t$ and $\underline{s} \equiv s$, or $\underline{t},\underline{s},t,s$ are pairwise distinct.

If $\underline{t} \equiv t$, $\underline{s} \equiv s$ then Tree($\underline{A}$) is $\underline{\Sigma} \cup \{t<s\}, \varepsilon \supset t<s$, and $\underline{T}$ is taken to be $\underline{\Sigma} \cup \{t<s,t'<s'\}, \varepsilon' \supset t<s \Rightarrow \underline{\Sigma} \cup \{t<s\}, \varepsilon' \supset t'<s'$, which is successful by $(assmp_A)$ and (μ_A), and has one-expansion. This $\underline{T}$ is longer but it still has depth 0.

If $\underline{t},t,\underline{s},s$ are pairwise distinct then Tree($\underline{A}$) is $\underline{\Sigma} \cup \{\underline{t}<\underline{s},t<s\}, \varepsilon \supset \underline{t}<\underline{s}$, and $\underline{T}$ is taken to be $\underline{\Sigma} \cup \{\underline{t}<\underline{s},t<s\}$, $\varepsilon' \supset \underline{t}<\underline{s}$ which is successful by $(assmp_A)$, has one-expansion, and has depth 0.

For the case (μ_A) we must have, by one-expansion, $\underline{t},\underline{s},t,s$ pairwise distinct. Then Tree($\underline{A}$) has the shape:

$\underline{\Sigma} \cup \{\underline{t}<\underline{s},t<s\}$, $\varepsilon \supset \varepsilon(\underline{t})<\varepsilon(\underline{s}) \Rightarrow \underline{\Sigma} \cup \{t<s\}, \varepsilon \supset \underline{t}<\underline{s}$

with $\underline{t},\underline{s} \in Dom(\varepsilon)$. Now $\sigma(\underline{t}) \equiv \underline{t}$, $\sigma(\underline{s}) \equiv \underline{s}$, and since $\underline{t},\underline{s} \in Dom(\varepsilon')$ we have $\varepsilon'(\underline{t})=\sigma(\varepsilon(\underline{t}))$, $\varepsilon'(\underline{s})=\sigma(\varepsilon(\underline{s}))$. The tree $\underline{T}$ is then chosen with the shape:

$\underline{\Sigma} \cup \{\underline{t}<\underline{s},t<s\}$, $\varepsilon' \supset \sigma(\varepsilon(\underline{t}))<\sigma(\varepsilon(\underline{s})) \Rightarrow \underline{\Sigma} \cup \{t<s\}, \varepsilon' \supset \underline{t}<\underline{s}$

hence preserving success and depth by (μ_A) and the induction hypothesis. One-expansion is preserved because, by induction hypothesis, $\underline{T}$ expands the same variables in (μ_A) node as Tree($\underline{A}$), which has one-expansion; except that t',s' are expanded in the $(assmp_A)$ case, but in the present situation $\underline{t},\underline{s},t',s'$ are distinct.

The other cases do not pose difficulties. One-expansion for the $(\rightarrow_A)$ case follows from one-expansion of the two branches, since one-expansion is defined pathwise.

Hence we can apply the induction hypothesis to (B), obtaining:

$\vdash_R \Sigma \cup \{t<s\} \supset \langle \sigma t_2,\varepsilon'\rangle < \langle \sigma s_2,\varepsilon'\rangle$

Then, by the equivalences $\langle \sigma t_2,\varepsilon'\rangle =_R \langle t_2,\varepsilon \backslash t\rangle$, $\langle \sigma s_2,\varepsilon'\rangle =_R \langle s_2,\varepsilon \backslash s\rangle$, and (eq_R), $(trans_R)$:

$\vdash_R \Sigma \cup \{t<s\} \supset \langle t_2,\varepsilon \backslash t\rangle < \langle s_2,\varepsilon \backslash s\rangle$

By a similar argument on $\Sigma \cup \{t<s\}, \varepsilon \supset s_1<t_1$ we obtain:

$\vdash_R \Sigma \cup \{t<s\} \supset \langle s_1,\varepsilon \backslash s\rangle < \langle t_1,\varepsilon \backslash t\rangle$

Finally, by $(\rightarrow_R)$:

$\Rightarrow \Sigma \cup \{t<s\} \supset \langle t_1,\varepsilon \backslash t\rangle \rightarrow \langle t_2,\varepsilon \backslash t\rangle < \langle s_1,\varepsilon \backslash s\rangle \rightarrow \langle s_2,\varepsilon \backslash s\rangle$
$\Leftrightarrow \Sigma \cup \{t<s\} \supset \langle \varepsilon(t),\varepsilon \backslash t\rangle < \langle \varepsilon(s),\varepsilon \backslash s\rangle$
$\Rightarrow \Sigma \supset \mu t.\langle \varepsilon(t),\varepsilon \backslash t\rangle < \mu s.\langle \varepsilon(s),\varepsilon \backslash s\rangle$ (μ_R)
$\Leftrightarrow \Sigma \supset \langle t,\varepsilon\rangle < \langle s,\varepsilon\rangle$ □

4.4.3 Theorem *(Completeness of the subtyping rules)*
If $\alpha<_T\beta$ then $\alpha<_R\beta$.

Proof

If $\alpha<_T\beta$ then $\alpha<_A\beta$ by completeness of the algorithm (3.3.7). Consider the corresponding successful execution tree and apply the lockstep recursion lemma 4.4.1, obtaining a tree for $\alpha'<_A\beta'$ with $\alpha=_T\alpha'$ and $\beta=_T\beta'$. By lemma 4.4.2 we can now extract from the new execution tree a proof of $\alpha'<_R\beta'$. Applying the completeness of the rules for type equivalence we conclude $\alpha=_R\alpha'$ and $\beta=_R\beta'$. Finally we derive $\alpha<_R\beta$ by (eq_R) and $(trans_R)$. □

5. A Per Model

We sketch the main features of a model described in [Amadio 89] (see also [Cardone Coppo 89] for a related work) based on *complete uniform pers* over a D_∞ λ-model [Scott 72].

Per (partial equivalence relation) models provide an interpretation of subtyping as set-theoretic containment of the relations [Bruce Longo 88]. In addition, these structures have very interesting categorical properties

(in particular cartesian closure and interpretation of second-order quantification as intersection, see [Hyland 89]) that entail a satisfying interpretation of higher-order typed λ-calculi. The particular class of pers considered here preserves the previous properties while providing a solution of recursive domain equations up to equality. This result is obtained by an application of Banach's theorem on the uniqueness of the fixpoint of a contractive operator over a complete metric space.

5.1 Realizability Structure

Consider the functor $G(D) \triangleq D^D + D\times D + At$ defined in the category of complete partial orders (cpo's) and embedding-projection pairs (hereafter called *pairs*). The cpo At is a collection of atomic values, and + is the coalesced sum. The morphism part of G is standard.

The cpo D_∞ is the initial fixpoint of the functor G, that is the colimit of the following ω-diagram:

$$D_0 \triangleq O \quad \text{(the cpo with one element)}$$
$$D_{n+1} \triangleq D_n^{D_n} + D_n\times D_n + At = G(D_n)$$

with uniquely determined pairs :

$(i_{n,n+1}, j_{n+1,n}) : D_n \to D_{n+1}$.

Let (i_n, j_n) be the pair between D_n and D_∞. Let $e_n \triangleq i_n(j_n(e))$ for $e\in D_\infty$. The cpo's $D_\infty^{D_\infty}$ and $D_\infty\times D_\infty$ are projected into D_∞ by means of the pairs: (i, j) and ([,], p). The operation of application on D_∞ is defined as usual as: $fd \triangleq j(f)(d)$.

5.2 Complete Uniform Pers

A per A over D_∞ is complete and uniform[4] (henceforth cuper) iff

(1) $(\perp_{D_\infty}, \perp_{D_\infty}) \in A$
($\perp_{D_\infty}$ is the least element of the cpo D_∞)
(2) If $X\subseteq A$ is directed in $D_\infty\times D_\infty$ then $\sqcup X \in A$
(3) If $(e,e') \in A$ then $\forall n.\ (e_n, e'_n) \in A$

We will consider the *full* subcategory of complete and uniform pers, therefore the *morphisms* are defined as usual as:

$$\text{cuper}[A, B] \triangleq$$
$$\{f: D_\infty/A \to D_\infty/B \mid$$
$$\exists \phi\in D_\infty.\forall d\in D_\infty.\ (d,d)\in A \Rightarrow \phi d \in f([d]_A)\}$$
$$\text{where } [d]_A \triangleq \{e\in D_\infty \mid (d, e)\in A\},$$
$$\text{and } D_\infty/A \triangleq \{[d]_A \mid (d, d)\in A\}$$

Let $A_{\mid n} \triangleq A \cap i_n(D_n)\times i_n(D_n)$. Given A, B cupers we can define as for ideals (see [MacQueen et al. 86]):

closeness:
$c(A,B) \triangleq \infty$, if $A=B$; $\max\{n \mid A_{\mid n}=B_{\mid n}\}$, o.w.
distance:
$d(A,B) \triangleq 0$, if $c(A,B) = \infty$; $2^{-c(A,B)}$, o.w.

5.2.1 Subtype Interpretation

Following [Cardelli 88] and [Bruce Longo 88] we say that the cuper A is a subtype of the cuper B iff $A\subseteq B$. This is easily shown to correspond to the existence of a unique map in the category that is *realized* by the identity. Such maps play the role of *coercions* from A to B.

[4] A term suggested by M. Abadi and G. Plotkin.

5.2.2 Type Interpretation

A *type environment* η is a map from type variables to cupers: η: Tvar→cuper. A *type interpretation* of a type α in an environment η is written as $[\![\alpha]\!]\eta$.

In view of the interpretation of subtyping, the interpretation of type variables and type constants is naturally given as follows:

$$[\![\perp]\!]\eta \triangleq \{(\perp_{D_\infty}, \perp_{D_\infty})\}$$
$$[\![\top]\!]\eta \triangleq D_\infty\times D_\infty \equiv \text{Top}$$
$$[\![t]\!]\eta \triangleq \eta(t)$$

As we already mentioned, cuper is a cartesian closed category. In particular, given A, B cupers the *exponent* B^A is defined as follows:

$$(f, g) \in B^A \iff \forall\, d, e\ (d, e) \in A \Rightarrow (fd, ge) \in B$$

This interpretation of the arrow is sometime referred to as *simple*.

In general, every object exp(A, B) *isomorphic* to the simple interpretation will enjoy the same categorical properties. Therefore, we assume exp is a binary operator on cupers satisfying:

$$\exp(A, B) \cong B^A$$

However, not any choice will be satisfying from our point of view. In order to complete the interpretation we need two more properties of the operator exp, namely, *contractiveness* and *(anti-)monotonicity*.

5.2.3 Contractiveness

The set of cupers endowed with the metric d is a *complete metric space*. We require that the behavior of exp at level n+1 is determined by the value of the arguments up to level n:

$$\exp(A, B)_{\mid n+1} = \exp(A_{\mid n}, B_{\mid n})_{\mid n+1}$$

Under this condition the exponentiation operator is *contractive* on the space (cuper, d) as it satisfies the following property:

$$A_{\mid n}=A'_{\mid n},\ B_{\mid n}=B'_{\mid n} \Rightarrow$$
$$\exp(A, B)_{\mid n+1} = \exp(A', B')_{\mid n+1}$$

It turns out that every definable type operator is either contractive or the identity, and therefore admits a least fixpoint. The type-interpretation w.r.t. a contractive exponent exp(A, B) is completed as follows:
$[\![\alpha\to\beta]\!]\eta \triangleq \exp([\![\alpha]\!]\eta, [\![\beta]\!]\eta)$ $[\![\mu t.\alpha]\!]\eta \triangleq \text{Lfp}(\lambda A.[\![\alpha]\!]\eta_{[A/t]})$
(Lfp≡least fixpoint).

5.2.4 Soundness of the (→) subtyping rule

In order to have a sound interpretation of the (→) rule in 2.2 it is convenient that the operator exp satisfies the following additional condition:

$$A'\subseteq A,\ B\subseteq B' \Rightarrow \exp(A, B) \subseteq \exp(A', B')$$

Proviso

We write $\models \alpha<\beta$ iff, given any binary operator exp(A,B) satisfying the conditions of *isomorphism with the simple semantics* (5.2.2), *contractiveness* (5.2.3), and *(anti-)monotonicity* (5.2.4), the resulting type-interpretation $[\![\ ]\!]$ satisfies $[\![\alpha]\!]\eta \subseteq [\![\beta]\!]\eta$ for any η: Tvar→cuper.

We also write $\models \Gamma \supset \alpha<\beta$. As usual this means: $\forall\eta.\ (\eta\models \Gamma \Rightarrow \eta\models \alpha<\beta)$.

5.2.5 Theorem *(Soundness of the tree ordering w.r.t. the model)*

Given α, β types, if $\alpha<_T\beta$ then $\models \alpha<\beta$.

Proof (sketch)

Given a per A we define its *completion* cmpl(A) as the least cuper that contains A:

$$\mathrm{cmpl}(A) \triangleq \bigcap \{B \text{ cuper} \mid A\subseteq B\}$$

Given a tree A in Tree(L) we define its interpretation as the completion of the set-theoretic union of the interpretations of its syntactic approximants:

$$[\![A]\!]\eta \triangleq \mathrm{cmpl}(\cup_{k<\omega}[\![A_{\mid k}]\!]\eta)$$

It is easy to observe that $\{[\![A_{\mid k}]\!]\eta \mid k<\omega\}$ is a growing chain of cupers.

Now we need the following fact (see [Amadio 89]):

$$\forall n,\alpha.\ \exists N.\ \forall k\geq N.\ [\![(\alpha)_n]\!]\eta = [\![(\alpha_{\mid k})_n]\!]\eta$$

where by definition $[\![(\beta)_n]\!]\eta \triangleq [\![\beta]\!]\eta \cap i_n(D_n)\times i_n(D_n)$.

In other words, if we are interested in the interpretation of the type α up to the n-th level of the construction of D_∞, it is enough to unfold α up to a certain level N and just consider the interpretation of this *finite* part of the associated tree expansion.

Next we use the fact that $[\![\alpha]\!]\eta = \mathrm{cmpl}(\cup_{n<\omega}[\![(\alpha)_n]\!]\eta)$. From this we can conclude $[\![\alpha]\!]\eta \subseteq [\![T\alpha]\!]\eta$.

Vice versa observe that $\forall k.\ [\![\alpha_{\mid k}]\!]\eta \subseteq [\![\alpha]\!]\eta$. Hence $[\![\alpha]\!]\eta = [\![T\alpha]\!]\eta$.

Finally, $T\alpha <_\infty T\beta \Rightarrow \forall k.\ (\alpha_{\mid k} < \beta_{\mid k}) \Rightarrow \forall k.\ [\![\alpha_{\mid k}]\!]\eta \subseteq [\![\beta_{\mid k}]\!]\eta \Rightarrow [\![\alpha]\!]\eta \subseteq [\![\beta]\!]\eta$. □

5.2.6 Proposition *(Soundness of the rule ordering w.r.t. the model)*

If $\vdash_R \Gamma \supset \alpha<\beta$ then $\models \Gamma \supset \alpha<\beta$.

Proof

For the soundness of the type equivalence rules (4.1) one observes that the contractiveness of α in t is a sufficient (and necessary) condition to enforce the contractiveness of the following functional on the space $\mathrm{cuper}_{D_\infty}$ (5.2):

$$G_{\alpha,\eta,t}(A) \triangleq [\![\alpha]\!]\eta[A/t] \qquad A\in \mathrm{cuper}_{D_\infty}$$

As for the subtyping rules (4.3) the problem is to check the soundness of (μ_R). Suppose $\eta\models \Gamma$. By hypothesis we have:

$$\forall A,B \text{ cuper}.\ A\subseteq B \Rightarrow G_\alpha(A) \triangleq [\![\alpha]\!]\eta[A/t] \subseteq [\![\beta]\!]\eta[B/t] \triangleq G_\beta(B)$$

Therefore we have: $\forall n.\ G_\alpha{}^n(\mathrm{Bot}) \subseteq G_\beta{}^n(\mathrm{Bot})$, where $\mathrm{Bot} = \{(\perp_{D_\infty}, \perp_{D_\infty})\}$.

It can be proved (see [Amadio 89]) that for any type γ:

$$[\![(\mu t.\gamma)_n]\!]\eta \triangleq [\![\mu t.\gamma]\!]\eta \cap D_n\times D_n = G_\gamma{}^n(\mathrm{Bot}) \cap D_n\times D_n$$

And from $[\![\mu t.\gamma]\!]\eta = \mathrm{cmpl}(\cup_{n<\omega}[\![(\mu t.\gamma)_n]\!]\eta)$ we have the thesis. □

5.3 Completeness of an F-interpretation

Define an *F-interpretation* of → (see [Scott 76]) as:

$$(B^A)_F \triangleq B^A \cap F^2 \cup \{(\perp, f), (f, \perp), (f, f)\}$$

where F is the embedding of the functional space $D_\infty{}^{D_\infty}$ into D_∞ and f is the embedding of a distinct symbol of At into D_∞. Roughly speaking $(B^A)_F$ is built from B^A by selecting among those elements that are "functions" in the underlying λ-model D_∞ and by attaching to ⊥ a label f.

We can *characterize the subtypings valid in every F-interpretation.* Add the following axioms to the subtyping system in 2.2:

$$(\Phi 1)\ \perp\to T< \alpha\to T \qquad (\Phi 2)^5\ \ \alpha\to\beta< \gamma\to T$$

$\vdash_\Phi$ denotes formal derivability in this new system. Write $\alpha<_\Phi\beta$ iff $\vdash_\Phi \alpha<\beta$.

Next, extend the preorder $<_\Phi$ to recursive types by defining an ordering $<_{\Phi\infty}$ on trees as: $A <_{\Phi\infty} B$ iff $\forall k.\ (A_{\mid k}<_\Phi B_{\mid k})$. Of course $\alpha<_{\Phi_T}\beta$ iff $T\alpha<_{\Phi\infty}T\beta$.

Let $\models_F \alpha<\beta$ mean that for any F-interpretation $[\![\ ]\!]$ and for any type environment η: $[\![\alpha]\!]\eta \subseteq [\![\beta]\!]\eta$. We build and F-interpretation $[\![\ ]\!]'$ and a type environment η' with the property that, e.g., for non-recursive types α and β: $[\![\alpha]\!]'\eta' \subseteq [\![\beta]\!]'\eta'$ iff $\alpha<_\Phi\beta$. Then, given recursive types α and β we can prove:

$$\models_F \alpha<\beta \text{ iff } \alpha<_{\Phi_T}\beta.$$

6. Coercions

Coercions and subtyping are closely related topics; see for example [Amadio 90], [Breazu-Tannen et al. 89]. We now show that the standard coercions $c_{\alpha,\beta}$ between two types $\alpha<\beta$ are definable in an extension of the basic calculus. This can be interpreted as saying that subtyping does not add any expressive power to such calculus (only convenience).

Then we show that the coercions implicit in a calculus with subsumption can be automatically synthesized. This fact is related to an algorithm for inferring the minimum type of a term.

6.1 Definability.

In this section we show how to associate with each successful execution tree a λ-term whose denotation in the model is a coercion, that is, the unique map between the corresponding types that is realized by the identity.

6.1.1 Building the λ-term.

We can show that if we consider types up to tree equivalence, $=_T$, then for every initial goal $\Sigma,\varepsilon \supset t<s$ such that $\vdash_A \Sigma,\varepsilon\supset t<s$ there is a term $M(x_1, \ldots, x_n) : \langle t\to s,\varepsilon\rangle$ where $\Sigma = \{t_1<s_1, \ldots, t_n<s_n\}$ and x_i (i=1,..,n) are the free variables of M of type $\langle t_i\to s_i,\varepsilon\rangle$.

[5] We need (Φ2) to have transitivity as a derived rule.

For the sake of readability the type labels on bound variables and on the fold and unfold constants are often omitted.

We recall that it is possible to define a *fixpoint combinator* as follows:
$\mathbf{Y} \equiv \lambda f^{\alpha\to\alpha}.(\lambda x^{\mu t.t\to\alpha}.\ f((\text{unfold } x)x))\ (\text{fold}(\lambda x^{\mu t.t\to\alpha}.\ f((\text{unfold } x)x))) : (\alpha\to\alpha)\to\alpha$.

Proceed by induction on the structure of the execution tree (see 3.2.1). We refer to 3.1.6 for the properties 1..6, of the translation $\langle -, - \rangle$:

Case (assmp) $x^{\langle t\to s,\varepsilon\rangle}$.

Case ($\bot$) $\lambda x^{\bot}.\ \mathbf{Y}(\lambda x^{\langle\beta,\varepsilon\rangle}.\ x) : \langle\bot\to\beta,\varepsilon\rangle =_T \bot\to\langle\beta,\varepsilon\rangle$ by 1,5.

Case ($\top$) $\lambda x^{\langle\alpha,\varepsilon\rangle}.\ \mathbf{Y}(\lambda x^{\top}.\ x) : \langle\alpha\to\top,\varepsilon\rangle =_T \langle\alpha,\varepsilon\rangle\to\top$ by 2,5.

Case (var) $\lambda x^{a}.\ x : \langle a\to a,\varepsilon\rangle =_T a\to a$ by 3,5.

Case ($\to$) $\lambda f^{\langle\alpha\to\beta,\varepsilon\rangle}.\ \lambda x^{\langle\alpha',\ \varepsilon\rangle}.\ M_2(f(M_1(x))) :$
$\langle(\alpha\to\beta)\to(\alpha'\to\beta'),\varepsilon\rangle$ by 5. Where by induction hypothesis $M_2 : \langle\beta\to\beta',\varepsilon\rangle$ and $M_1 : \langle\alpha'\to\alpha,\varepsilon\rangle$.

Case (μ) by induction hypothesis we have $M(x^{\langle t\to s,\varepsilon\rangle}) : \langle\varepsilon(t)\to\varepsilon(s),\ \varepsilon\rangle$; by 4, 5 $\langle t\to s,\varepsilon\rangle =_T \langle\varepsilon(t)\to\varepsilon(s),\ \varepsilon\rangle$ therefore we can type a term:
$\mathbf{Y}(\lambda y^{\langle\varepsilon(t)\to\varepsilon(s),\ \varepsilon\rangle}.\ M(y)) : \langle\varepsilon(t)\to\varepsilon(s),\ \varepsilon\rangle$

6.1.2 Proposition *(Coercions are definable)*
Let $\alpha, \beta \in$ Type and suppose $\alpha <_A \beta$. Let M be the term associated in 6.1.1 with the execution tree of $\emptyset, E\alpha \cup E\beta \supset \alpha^* < \beta^*$. Then the denotation of the term in the model is the unique coercion map from the interpretation of α to the interpretation of β.

Proof.
Since we have not given the term interpretation explicitly (see [Amadio 89]), we can only sketch an idea of the proof.

In the first place we need some facts about the interpretation of terms:
(a) By erasing the type information and the constants fold, unfold from a typed term M, we obtain an untyped λ-term *er*(M). We denote these untyped λ-terms with P, Q, ... It is a basic property of these interpretations that the interpretation of *er*(M) gives a representative for the equivalence class that corresponds to the interpretation of M. We shortly refer to this fact by saying that *er*(M) is a realizer for M.
(b) Showing that the interpretation of M is a coercion from α to β means proving that the identity map, id, is a realizer for M. Equivalently id and *er*(M) are equivalent in $\alpha\to\beta$. Note that here and in the following for the sake of readability we simply refer to syntactic objects but we really intend to speak of their denotations in the model.
(c) The realizer for **Y** is an element Fix with functionality: $\lambda g.\ \sqcup g^n(\bot_{D_\infty})$.

In order to prove the theorem by induction on the structure of the execution tree one needs to generalize somewhat.

In the first place one observes that if in the execution tree of $\emptyset, \varepsilon \supset t < s$ we never use (assmp) then the interpretation of the associated term $M : \langle t\to s,\varepsilon\rangle$ is a coercion in $\langle t\to s,\varepsilon\rangle$.

However, this is not enough to make the induction go through in the case where the term $M(x^{\Sigma})$ really depends on the assumption variable. One has to observe that $M(x^{\Sigma})$ also enjoys a property of *contractiveness*.

Let us suppose that (μ) is the last rule applied. By construction assume we have a term M(x) that is a functional from coercions to coercions. We would like to show that $\mathbf{Y}(\lambda x.M(x))$ is still a coercion.

Observe that after a (μ) rule we always have a ($\to$) rule. Therefore the term M(x) has the structure $\lambda f.\lambda y.M_2(x)(f(M_1(x)y))$.

Now observe that a realizer for $\mathbf{Y}(\lambda x.M(x))$ will be something like $\sqcup g^n(\bot_{D_\infty})$ for $g = \lambda x.\lambda f.\lambda y.P_2(x)(f(P_1(x)y))$ where P_i is a realizer for M_i (i=1,2). We have to show that this realizer is equivalent to id in a type with the structure $C \equiv (A \Rightarrow B) \Rightarrow (A' \Rightarrow B')$, where $A \Rightarrow B \equiv \exp(A, B)$. Since the type is a complete per, it will be enough to show that for each n $g^n(\bot_{D_\infty})$ is equivalent to id in the appropriate type.

To do this we need a last remark, Denote with $A_{|n}$ the approximation at the n-th level of the cuper A as in 5.2. One observes that if $(P, id) \in C_{|n}$ then $(g(P), id) \in C_{|n+1}$. This follows easily from the structure of g and the assumption (5.2.3). Hence we have $\forall n.\ (g^n(\bot_{D_\infty}), id) \in C_{|n}$ that implies $(\sqcup g^n(\bot_{D_\infty}), id) \in C$. □

6.2 Inference
Let $\lambda^{\to\mu}$ be the calculus in section 2. Given a term in $\lambda^{\to\mu}$, possibly not typeable, we are interested in the problem of determining if it can be *well-typed modulo the insertion of appropriate coercions.*

We refer to this problem as *coercion inference.* We will define a simple algorithm that, given a term M, succeeds exactly when M is typeable modulo the insertion of coercions. In this case the algorithm returns the *least type* among the types that can be assigned to M.

A similar problem was solved in [Amadio 89b] for a second-order lambda calculus with records, and in [Curien Ghelli 90] for a second-order lambda calculus including a form of bounded quantification.

All these results rely on the *structural properties of the subtype relation* that are stated, in this case, as Proposition 6.2.4.

Notation.
In this section $\alpha \lesssim \beta$ and $\alpha \simeq \beta$ are shorthands for $T\alpha <_\infty T\beta$ and $T\alpha = T\beta$.

6.2.1 Typing modulo coercions
We can formalize the idea of *typing modulo coercions* in two ways:

(a) Subsumption. Add to the typing system in 2.1 and 2.2 the following rule based on the tree order $<_\infty$. The version based on $<_{fin}$ is often referred to as Subsumption:
(Sub_∞) $\quad M: \alpha,\ \alpha \lesssim \beta \Rightarrow M: \beta$
We denote formal derivability in this system with $\vdash_{Sub}$.

(b) Explicit Coercions. Extend the term language with a collection of constants $\{c_{\alpha,\beta} \mid \alpha, \beta \text{ types}\}$ and add to the typing system in 2.2 the following rule:

(ExpCoer$_\infty$) $\quad M: \alpha,\ \alpha \precsim \beta \Rightarrow (c_{\alpha,\beta} M): \beta$

Denote formal derivability in this new system with $\vdash_c$, and denote the corresponding term language with $\lambda^{\to\mu c}$. Moreover, denote with er_c (mnemonic for erase coercions) the obvious function that takes a term in $\lambda^{\to\mu c}$, erases all the constants $c_{\alpha,\beta}$, and returns a term in $\lambda^{\to\mu}$.

The use of these rules is justified by the finitary axiomatization of $\precsim$ given in section 4.

Note that in both these systems the $(\text{fold}_{\mu t.\alpha} M)$ and $(\text{unfold}_{\mu t.\alpha} M)$ terms become redundant.

6.2.2 Definition *(coercion inference)*

We define inductively on the structure of the term M in $\lambda^{\to\mu}$ a function[6]

CI: $(\lambda^{\to\mu}) \to (\lambda^{\to\mu c} \cup \{\text{FAIL}\})$ (CI=coercion inference) that either fails or returns a well-typed term N in $\lambda^{\to\mu c}$ such that $er_c(N) \equiv M$.

It is intended that the clauses (fold), (unfold) have priority on the clause (apl).

(var) $\quad CI(x^\alpha) \triangleq x^\alpha$

(abs) $\quad CI(\lambda x^\alpha.M) \triangleq$
if $CI(M) : \beta$ then $\lambda x^\alpha.CI(M)$ else FAIL

(apl) $\quad CI(MN) \triangleq$
if $CI(M) : \alpha'$ and $CI(N): \gamma$ then
if $\alpha' \simeq \alpha \to \beta$ and $\gamma \precsim \alpha$
then $(c_{\alpha', \alpha\to\beta}\ CI(M))(c_{\gamma,\alpha}\ CI(N))$
else if $\alpha' \simeq \bot$
then $(c_{\alpha', \alpha\to\bot}\ CI(M))\ CI(N)$
else FAIL
else FAIL

(fold) $\quad CI(\text{fold}_{\mu t.\alpha} M) \triangleq$
if $CI(M): \beta$ and $\beta \precsim \mu t.\alpha$
then $\text{fold}_{\mu t.\alpha}(c_{\beta,[\mu t.\alpha/t]\alpha} CI(M))$
else FAIL

(unfold) $\quad CI(\text{unfold}_{\mu t.\alpha} M) \triangleq$
if $CI(M): \beta$ and $\beta \precsim \mu t.\alpha$
then $\text{unfold}_{\mu t.\alpha}(c_{\beta,\mu t.\alpha} CI(M))$
else FAIL □

Clearly CI can also be used to define an inference algorithm for $\vdash_{Sub}$; just consider the type of the term synthesized by CI. We prove in 6.2.5 that this algorithm computes the minimal type of a term (if any). To achieve this result we need the following simple properties.

6.2.3 Proposition

Let M be a term in $\lambda^{\to\mu}$ then:

(1) $\vdash_{Sub} M : \alpha$ iff for some N: $\vdash_c N : \alpha$ and $er_c(N) \equiv M$.

(2) If $CI(M): \beta$ then $er_c(CI(M)) \equiv M$.

Proof

(1) Every introduction of an explicit coercion corresponds to an application of subsumption and vice versa.

(2) By induction on the definition of CI. □

6.2.4 Proposition *(Structural subtyping)*

Let $\alpha, \beta, \ldots$ be recursive types then:

(1) If $\alpha \precsim \beta_1 \to \beta_2$ then either $\alpha \simeq \bot$ or $\alpha \simeq \alpha_1 \to \alpha_2$, $\beta_1 \precsim \alpha_1$, and $\alpha_2 \precsim \beta_2$.

(2) If $\alpha_1 \to \alpha_2 \precsim \beta$ then either $\beta \simeq \top$ or $\beta \simeq \beta_1 \to \beta_2$, $\beta_1 \precsim \alpha_1$, and $\alpha_2 \precsim \beta_2$.

Proof

(1) α can be rewritten, by unfolding, to an equivalent type of the shape $\bot$, $\top$, t or $\alpha_1 \to \alpha_2$. The definition of the tree ordering and the hypothesis $\alpha \precsim \beta_1 \to \beta_2$ lead to the conclusion by a simple case analysis.

(2) Analogous. □

6.2.5 Theorem *(Terms have a least type)*

Let M be a term in $\lambda^{\to\mu}$ then $\vdash_{Sub} M : \alpha$ implies $CI(M):\beta$ and $\beta \precsim \alpha$.

Proof

By induction on the structure of M.

C(N) is a meta-notation for $c\alpha_{n-1,\alpha_n}(\ldots(c\alpha_{1,\alpha_2} N)\ldots)$, where: N: α_1, $n \geq 1$, $\alpha_i \precsim \alpha_{i+1}$.

By virtue of 6.2.3.(1) we may equivalently assume the existence of a well-typed term N in $\lambda^{\to\mu c}$ such that $er_c(N) \equiv M$.

Observe the crucial role of property 6.2.4 in proving the rather surprising fact that the algorithm is *complete* in the sense just stated above.

Case $M \equiv x^\beta$.

If $er_c(N) \equiv x^\beta$ then $N \equiv C\ x^\beta : \alpha$ and $\beta \precsim \alpha$. On the other hand $CI(x^\beta) \equiv x^\beta:\beta$.

Case $M \equiv (\lambda x\alpha.M')$.

If $er_c(N) \equiv \lambda x^\alpha.M'$ then $N \equiv C\ (\lambda x^\alpha.N')$: γ, $er_c(N') \equiv M'$, and $N':\beta'$. By induction hypothesis $CI(M') : \beta$ and $\beta \precsim \beta'$, hence by definition. $CI(\lambda x^\alpha.M') : \alpha \to \beta$. Note that $\alpha \to \beta' \precsim \gamma$ by definition of N and this implies (by 6.2.4) either $\gamma \simeq \top$ (and in this case we are done as $\alpha \to \beta \precsim \top$) or $\gamma \simeq \gamma_1 \to \gamma_2$, $\gamma_1 \precsim \alpha$ and $\beta' \precsim \gamma_2$. In the latter case $\beta \precsim \beta' \precsim \gamma_2$ implies $\alpha \to \beta \precsim \gamma$.

Case $M \equiv (M_1 M_2)$.

If $er_c(N) \equiv M_1 M_2$ then $N \equiv C(N_1 N_2)$: γ, $er_c(N_i) \equiv M_i$ $i=1,2$, N_1: $\gamma_1 \to \gamma_2$, N_2:γ_1. By induction hypothesis $CI(M_i) : \beta_i$ $i=1, 2$, $\beta_1 \precsim \gamma_1 \to \gamma_2$ and $\beta_2 \precsim \gamma_1$. From (6.2.4) follows that $\beta_1 \simeq \bot$ or $\beta_1 \simeq \beta_1' \to \beta_1''$, $\gamma_1 \precsim \beta_1'$, $\beta_1'' \precsim \gamma_2$. In the first case $CI(M_1 M_2) : \bot$ and we are done. In the second $CI(M_1 M_2) : \beta_1''$ as $\beta_2 \precsim \gamma_1 \precsim \beta_1'$. Finally observe: $\beta_1'' \precsim \gamma_2 \precsim \gamma$.

Case $M \equiv (\text{fold}_{\mu t.\alpha} M')$.

If $er_c(N) \equiv \text{fold}_{\mu t.\alpha} M'$ then $N \equiv C(\text{fold } N')$: γ, $er_c(N') \equiv M'$, N': $[\mu t.\alpha/t]\alpha \equiv \gamma'$, $\gamma' \precsim \gamma$. By induction hypothesis $CI(M')$: β', $\beta' \precsim \gamma'$. Hence by definition, $CI(\text{fold } M')$: $\mu t.\alpha$ and we have $\mu t.\alpha \simeq \gamma' \precsim \gamma$.

Case $M \equiv (\text{unfold}_{\mu t.\alpha} M')$.

Analogous. □

[6] Actually the following specification determines a *class* of algorithms that suffices for our purposes.

7. Conclusion

We have used a subtyping relation based on infinite trees as the central concept of our work. In our experience this relation has arisen naturally, giving insights about both the subtypings valid in certain per-models and the behavior of the Amber implementation. In fact we have shown that this relation can be used to characterize sound and complete theories for a certain class of per models and that it can be simply and efficiently implemented. We have also shown the soundness and completeness of certain rules and the definability of coercions within the calculus (modulo a strengthening of the notion of type equality). Finally, we have observed that the whole process of inferring coercions and minimal types can be automated.

In conclusion, let us consider the problem of the extension of our results.

The notions of tree expansion and finite approximation (section 3) can be easily adapted to larger languages, both with first-order type constructors like products, sums, records and variants, and with higher-order type constructors like second-order universal quantification (e.g. for Quest [Cardelli 89]). The important point is that the tree resulting from the expansion is regular. Under this assumption it seems possible to adapt algorithms and rules to obtain results of soundness and completeness (sections 3, 4).

About the relationship between the tree ordering and the model, we expect the extension of the soundness theorem (5.2) to be straightforward. On the other hand we expect technical problems from the completeness theorem (5.3) when introducing higher-order type constructors like second-order universal quantification.

The result on the definability of the coercions has already been obtained for several calculi with records, variants, and bounded quantification (but without recursion). It is a reassuring result that shows that the subtyping theory is in good harmony with the calculus.

The fact that terms have a least type has a clear impact on the implementation of the type-checker. This appears to be a very desirable property towards an automatic treatment of coercions. The result, at the present state of the art, clearly relies on the *structural properties* of the subtyping relation.

Finally, we observe that challenging extensions arise when dealing with non-ground collections of subtyping assumptions (see [Amadio90]). In this case much work remains to be done.

Acknowledgments

We would like to thank Martín Abadi for comments on an early draft, and Mario Coppo for discussions.

References

[Amadio 89] R. Amadio: **Recursion over realizability structures**, TR1/89 Dipartimento di Informatica, Università di Pisa, to appear in Info.&Comp. .

[Amadio 89b] R. Amadio: **Formal theories of inheritance for typed functional languages**, TR 28/89 Dipartimento di Informatica, Università di Pisa.

[Amadio 90] R. Amadio: **Typed equivalence, type assignment and type containment**, abstract in Proc. CTRS90, eds. Kaplan&Okada, Montreal, June 1990.

[Arnold Nivat 80] A.Arnold, M.Nivat: **The metric space of infinite trees. Algebraic and topological properties**, Fundamenta Informaticae III pp.445-476, 1980.

[Breazu-Tannen et al. 89] V. Breazu-Tannen, C. Gunter, A. Scedrov: **Denotational semantics for subtyping between recursive types**, Report MS-CIS 89 63, Logic of Computation 12, Dept of Computer & Information Science, University of Pennsylvania.

[Bruce Longo 88] K. Bruce, G. Longo: **A modest model of records, inheritance and bounded quantification**, IEEE-LICS 88, Edinburgh.

[Cardelli 86] L. Cardelli: **Amber**, in Combinators and Functional Programming Languages, Lecture Notes in Computer Science n. 242, Springer-Verlag, 1986.

[Cardelli 88] L. Cardelli: **A semantics of multiple inheritance**, Info.&Comp., 76, pp.138-164.

[Cardelli 89] L.Cardelli: **Typeful programming**, SRC Report #45, Digital Equipment Corporation, 1989.

[Cardelli Donahue Jordan Kalsow Nelson 89] L.Cardelli, J.Donahue, M.Jordan, B.Kalsow, G.Nelson: **The Modula-3 type system**, Proc. POPL'89.

[Cardelli Longo 90] L.Cardelli, G.Longo: **A semantic basis for Quest**, Proc. of LISP&FP'90, Nice.

[Cardone Coppo 89] F.Cardone, M.Coppo: **Type inference with recursive types: syntax and semantics**, Dipartimento di Informatica, Università di Torino.

[Courcelle 83] B.Courcelle: **Fundamental properties of infinite trees**, Theoretical Computer Science, 25, pp.95-169, 1983.

[Curien Ghelli 90] P.L. Curien, G. Ghelli: **Coherence of Subsumption**, CAAP 1990, København.

[Hyland 89] M. Hyland: **A small complete category**, APAL 40, 2, pp.135-165.

[MacQueen et al. 86] D. MacQueen , G. Plotkin, R. Sethi: **An ideal model for recursive polymorphic types**, Info.&Comp., 71, 1-2.

[Milner 84] R.Milner: **A complete inference system for a class of regular behaviours**, JCSS 28, pp. 439-466, 1984.

[Park 81] D.M.R.Park: **Concurrency and automata on infinite sequences**, Proc. 5th GI conference, LNCS 104 pp.167-183, Springer-Verlag, 1981.

[Salomaa 66] A. Salomaa: **Two complete systems for the algebra of regular events**, JACM 13,1, 1966.

[Scott 72] D. Scott : **Continuous lattices**, Toposes, Algebraic Geometry and Logic, Lawvere (ed.), LNM 274, pp.97-136, Springer-Verlag, 1972.

[Scott 76] D. Scott : **Data types as lattices**, SIAM J. of Comp., 5, pp.522-587.