Mathematical Structures in Computer Science, 5(1):113–125, 1995.

Also in Proc. POPL'93, pages 419–428.

Efficient Recursive Subtyping

Dexter Kozen*
kozen@cs.cornell.edu

Jens Palsberg†
palsberg@daimi.aau.dk

Michael I. Schwartzbach†
mis@daimi.aau.dk

Abstract

Subtyping in the presence of recursive types for the λ-calculus was studied by Amadio and Cardelli in 1991 [1]. In that paper they showed that the problem of deciding whether one recursive type is a subtype of another is decidable in exponential time.

In this paper we give an $O(n^2)$ algorithm. Our algorithm is based on a simplification of the definition of the subtype relation, which allows us to reduce the problem to the emptiness problem for a certain finite automaton with quadratically many states.

It is known that equality of recursive types and the covariant Böhm order can be decided efficiently by means of finite automata, since they are just language equality and language inclusion, respectively. Our results extend the automata-theoretic approach to handle orderings based on contravariance.

*Computer Science Department, Cornell University, Ithaca, New York 14853, USA.
†Computer Science Department, Aarhus University, 8000 Aarhus C, Denmark.

1 Introduction

Recursive types are present in most programming languages, since they provide a means of typing recursive functions and data structures. Subtyping is also present in many languages and is especially important in object-oriented languages as a means of typing functions in the presence of inheritance and late binding.

The unrestricted combination of recursion and subtyping, found for example in Amber [2] and Quest [4, 3], is of substantial pragmatic value. Since it does not depend on programmer-defined names, it allows the flexible typing of such constructs as data persistence and data migration.

The combination of recursive types and subtyping at an abstract level was studied by Amadio and Cardelli in 1991 [1]. They considered types for the λ-calculus generated by the following grammar, where v is a type variable:

$$t ::= v \mid \bot \mid \top \mid t_1 \to t_2 \mid \mu v.t$$

Intuitively, $\bot$ is a minimal type containing only the divergent computation; $\top$ is a maximal or universal type containing all values; $t_1 \to t_2$ is the usual function space; and $\mu v.t$ is a recursive type that satisfies the equation

$$\mu v.t \quad = \quad t[v/\mu v.t] \ ,$$

where $t[v/s]$ denotes the term t with s substituted for free occurrences of v (after renaming bound variables if necessary).

In Amadio and Cardelli's approach, types are understood as collections of values, and subtypes are subcollections. Thus, types are partially ordered by an inclusion relation $\leq$. It is postulated that $\bot \leq t \leq \top$ for any type t, and function spaces are ordered by the usual rule

$$s \to t \leq s' \to t' \quad \text{if and only if} \quad s' \leq s \text{ and } t \leq t' \ ,$$

i.e., $\to$ is covariant in the range and contravariant in the domain. This defines a partial order inductively on finite types, but not on recursive types.

Amadio and Cardelli showed how to extend the ordering to recursive types. Their definition involves a rule of the form

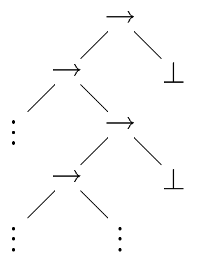
$$(v \leq v' \Rightarrow t \leq t') \quad \Rightarrow \quad (\mu v.t \leq \mu v'.t') \ ,$$

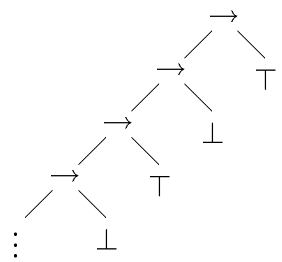
where v occurs only in t and v' occurs only in t'. In other words, if by assuming the inclusion of the recursion variables we can verify the inclusion of the bodies, then we can deduce the inclusion of the recursive types.

They also considered the standard representation of types as labeled trees, defined a partial order on infinite trees, and showed that it agrees with the type inclusion order. Their definition of the order on trees involves infinite sequences of finite approximations, where the approximations are obtained by truncating the trees at some finite level. The relation $\leq$ holds between two trees iff it holds between all their finite truncations.

For an illustration of the type and tree orderings, consider the following two types and their tree representations.

$\mu u.((u \to u) \to \bot)$ $\qquad\qquad$ $\mu v.((v \to \bot) \to \top)$





It can be shown using Amadio and Cardelli's type rules that the left type is included in the right. It is somewhat easier to see this for the corresponding trees: all level-k truncations are clearly ordered from left to right.

In order to automate type checking in the presence of subtypes and recursive types, the problem of deciding type inclusion is of paramount importance:

Given two types s and t, is $s \leq t$?

Amadio and Cardelli showed that this problem is decidable, but gave no complexity analysis. However, their algorithm involves the explicit construction of a binary tree of polynomial depth, thus is in some cases exponential. Their algorithm is based on a concrete representation of recursive types involving back-pointers to represent recursion.

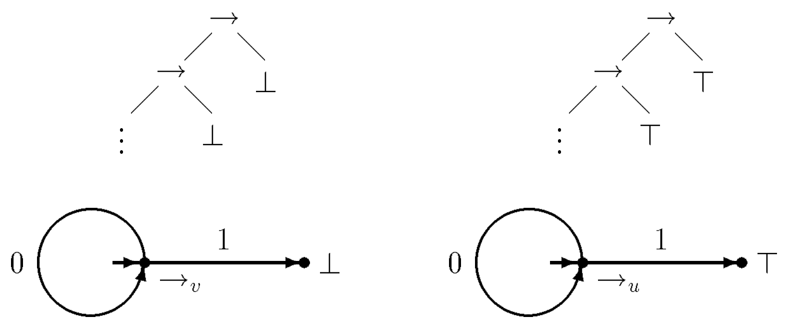
In this paper we show that the type inclusion problem is solvable in time $O(n^2)$. Our algorithm is based on a simplification of Amadio and Cardelli's definition of the subtype relation on trees; we show that the two definitions are equivalent. Our definition, which is a generalization of an order introduced by us in [7], intuitively says:

> Two trees are ordered if no common path detects a counterexample.

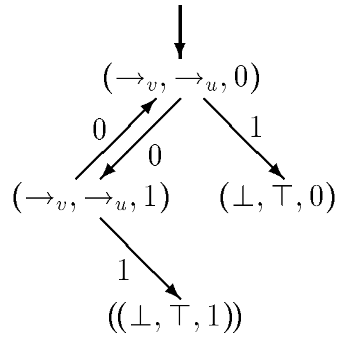
This allows us to reduce the problem to the emptiness problem for a certain finite automaton which accepts a language of counterexamples.

Our algorithm represents recursive types as so-called *term automata.* The automaton that detects counterexamples is then defined as a certain product of two term automata. For an illustration of this, consider the following two types, their tree representations, and the term automata for these trees.

$$\mu v.(v \to \bot) \;\not\leq\; \mu u.(u \to \top)$$



These two types are *not* in the subtype relation: consider for example their level-3 truncations. This can be detected by the following product automaton (we show only the reachable states):



The idea is that the accept states (marked with double parentheses) are those where the first component is *not* less than the second component in the ordering $\perp \leq \rightarrow \leq \top$. Because of contravariance, however, we use the third component to signal if the ordering should be reversed: 0 means "no" and 1 means "yes". The automaton above accepts the word 01, thus the level-3 truncations of the trees are not ordered.

The test for emptiness takes linear time in the size of the product automaton using depth first search. The size of the product automaton is the product of the sizes of the two term automata. Thus, our algorithm runs in $O(n^2)$ time.

It may be surprising that the inclusion of recursive types can be decided efficiently using finite automata. To quote Amadio and Cardelli [1]:

> The problem of equating recursive types ... can be related to well-known solvable problems, such as the equivalence of finite-state automata. However, the similar problem for subtyping has no well-known parallel.

Our results establish that the automata-theoretic approach is fruitful even in the presence of subtyping and contravariance. Further evidence is provided by the results of [7], which establish the first known polynomial time algorithm for a type inference problem studied by Thatte [10] and O'Keefe and Wand [9].

In the remainder of the paper we provide the definitions of term automata and labeled trees, prove that Amadio and Cardelli's tree ordering and ours agree, and give the details of our algorithm. We introduce term automata and labeled trees in a more general form than needed here; they may be useful in other contexts.

2 Terms

Here we give a general definition of (possibly infinite) terms over an arbitrary finite ranked alphabet Σ. Such terms are essentially labeled trees, which we represent as partial functions labeling strings over ω (the natural numbers) with elements of Σ. In our application, types are terms over the ranked alphabet $\{\bot, \rightarrow, \top\}$; finite types are finite terms and recursive types are regular terms.

Let Σ_n denote the set of elements of Σ of arity n. Let ω denote the set of natural numbers and let ω^* denote the set of finite-length strings over ω.

Definition 1 A *term* over Σ is a partial function

$$t : \omega^* \quad \rightarrow \quad \Sigma$$

with domain $\mathcal{D}(t)$ satisfying the following properties:

- $\mathcal{D}(t)$ is nonempty and prefix-closed;
- if $t(\alpha) \in \Sigma_n$, then $\{i \mid \alpha i \in \mathcal{D}(t)\} = \{0, 1, \ldots, n-1\}$.

The set of all terms is denoted T_Σ.

An element $\alpha \in \omega^*$ is a *leaf* of t if $\alpha \in \mathcal{D}(t)$ and α is not a proper prefix of any other element of $\mathcal{D}(t)$; equivalently, if $t(\alpha) \in \Sigma_0$. $\Box$

A term t is *finite* if its domain $\mathcal{D}(t)$ is a finite set. We denote the set of finite terms over Σ by F_Σ. A *path* in a term t is a maximal subset of $\mathcal{D}(t)$ linearly ordered by the prefix relation. By König's Lemma, a term is finite iff it has no infinite paths.

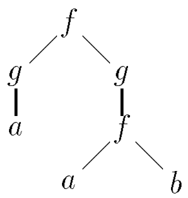
Definition 2 Let t be a term and $\alpha \in \omega^*$. Define the partial function $t\!\downarrow\!\alpha : \omega^* \rightarrow \Sigma$ by

$$t\!\downarrow\!\alpha(\beta) \quad = \quad t(\alpha\beta) \ .$$

If $t\!\downarrow\!\alpha$ has nonempty domain, then it is a term, and is called the *subterm of t at position α*. $\Box$

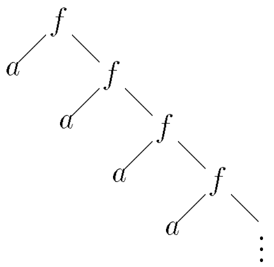
Definition 3 A term t is said to be *regular* if it has only finitely many distinct subterms; *i.e.*, if $\{t\!\downarrow\!\alpha \mid \alpha \in \omega^*\}$ is a finite set. The set of regular terms is denoted R_Σ. $\Box$

Example 4 Let $\Sigma = \{f, g, a, b\}$, where f, g, a, b have arities 2,1,0,0 respectively. The following picture represents a typical finite term t:



The leaves of t are the strings $00, 100, 101$ with $t(00) = t(100) = a$ and $t(101) = b$. The domain $\mathcal{D}(t)$ of t is the set of all prefixes of these strings, namely $\epsilon, 0, 1, 10$ in addition to those already mentioned, with $t(\epsilon) = t(10) = f$ and $t(0) = t(1) = g$.

The following picture represents a typical infinite regular term s:



The domain of s is the infinite regular set $1^* + 1^*0$, with $s(1^n0) = a$ and $s(1^n) = f$ for all $n \geq 0$. The leaves are the elements of the regular subset 1^*0. The term is regular because it has only two subterms, namely s itself and the singleton term a. $\square$

The sets T_Σ, F_Σ, and R_Σ become algebraic structures of signature Σ under the natural syntactic definition of the operators

$$f^{T_\Sigma} : T_\Sigma^n \;\; \rightarrow \;\; T_\Sigma$$

for each $f \in \Sigma_n$ given by

$$\begin{aligned} f^{T_\Sigma}(t_0, \ldots, t_{n-1})(i\alpha) &= t_i(\alpha), \quad 0 \leq i < n \\ f^{T_\Sigma}(t_0, \ldots, t_{n-1})(\epsilon) &= f \;. \end{aligned}$$

Then

$$\mathcal{D}(f^{T_\Sigma}(t_0, \ldots, t_{n-1})) \quad = \quad \{\epsilon\} \cup \bigcup_{i=0}^{n-1} \{i\alpha \mid \alpha \in \mathcal{D}(t_i)\} \ .$$

In particular, for $c \in \Sigma_0$, we have $c^{T_\Sigma}(\epsilon) = c$ and c^{T_Σ} undefined otherwise.

The following lemma establishes some elementary properties of these operators.

Lemma 5

(i) If $f \in \Sigma_n$ and $0 \leq i < n$, then $f^{T_\Sigma}(t_0, \ldots, t_{n-1}) \downarrow i = t_i$.

(ii) If $t(\epsilon) = f \in \Sigma_n$, then $t = f^{T_\Sigma}(t \downarrow 0, \ldots, t \downarrow (n-1))$.

(iii) $(t \downarrow \alpha) \downarrow \beta = t \downarrow \alpha\beta$.

(iv) The string α is a leaf of t iff $\mathcal{D}(t \downarrow \alpha) = \{\epsilon\}$.

Proof. All properties are immediate consequences of the definitions. □

3 Term Automata

Every regular term over a finite ranked alphabet Σ has a finite representation in terms of a special type of automaton called a *term automaton*.

Definition 6 Let Σ be a finite ranked alphabet. A *term automaton* over Σ is a tuple

$$\mathcal{M} \quad = \quad (Q,\ \Sigma,\ q_0,\ \delta,\ \ell)$$

where:

- Q is a finite set of *states*,
- $q_0 \in Q$ is the *start state*,
- $\delta : Q \times \omega \to Q$ is a partial function called the *transition function*, and
- $\ell : Q \to \Sigma$ is a (total) *labeling function*,

such that for any state $q \in Q$, if $\ell(q) \in \Sigma_n$ then

$$\{i \mid \delta(q,i) \text{ is defined}\} \quad = \quad \{0,1,\ldots,n-1\} \; .$$

We decorate Q, δ, *etc.* with the superscript $\mathcal{M}$ where necessary. □

Let $\mathcal{M}$ be a term automaton as in Definition 6. The partial function δ extends naturally to a partial function

$$\widehat{\delta} : Q \times \omega^* \to Q$$

inductively as follows:

$$\begin{aligned} \widehat{\delta}(q,\epsilon) &= q \\ \widehat{\delta}(q,\alpha i) &= \delta(\widehat{\delta}(q,\alpha),i) \; . \end{aligned}$$

For any $q \in Q$, the domain of the partial function $\lambda\alpha.\widehat{\delta}(q,\alpha)$ is nonempty (it always contains ϵ) and prefix-closed. Moreover, because of the condition on the existence of i-successors in Definition 6, the partial function

$$\lambda\alpha.\ell(\widehat{\delta}(q,\alpha))$$

is a term.

Definition 7 Let $\mathcal{M}$ be a term automaton. The term *represented by* $\mathcal{M}$ is the term

$$t_{\mathcal{M}} \quad = \quad \lambda\alpha.\ell(\widehat{\delta}(q_0,\alpha)) \; .$$

A term t is said to be *representable* if $t = t_{\mathcal{M}}$ for some $\mathcal{M}$. □

Intuitively, $t_{\mathcal{M}}(\alpha)$ is determined by starting in the start state q_0 and scanning the input α, following transitions of $\mathcal{M}$ as far as possible. If it is not possible to scan all of α because some i-transition along the way does not exist, then $t_{\mathcal{M}}(\alpha)$ is undefined. If on the other hand $\mathcal{M}$ scans the entire input α and ends up in state q, then $t_{\mathcal{M}}(\alpha) = \ell(q)$.

Lemma 8 *Let $t \in T_\Sigma$. The following are equivalent:*

(i) t is regular;

(ii) t is representable;

(iii) t is described by a finite set of equations involving the μ operator.

Proof. (i) $\Longrightarrow$ (ii) Suppose t has only finitely many subterms. Define

$$\begin{aligned} Q &= \{t{\downarrow}\alpha \mid \alpha \in \omega^*,\ \mathcal{D}(t{\downarrow}\alpha) \neq \emptyset\} \\ q_0 &= t \;=\; t{\downarrow}\epsilon \\ \ell(s) &= s(\epsilon) \\ \delta(s,i) &= \begin{cases} s{\downarrow}i\ , & \text{if } 0 \leq i \leq \text{arity}(\ell(s)) \\ \text{undefined}\ , & \text{otherwise} \end{cases} \end{aligned}$$

and let $\mathcal{M}$ be the automaton with these data. A straightforward inductive argument using Lemma 5 shows that

$$\widehat{\delta}(t,\alpha) = \begin{cases} t{\downarrow}\alpha\ , & \text{if } \mathcal{D}(t{\downarrow}\alpha) \neq \emptyset \\ \text{undefined}\ , & \text{otherwise} \end{cases}$$

thus

$$\begin{aligned} \ell(\widehat{\delta}(q_0,\alpha)) &= \ell(\widehat{\delta}(t,\alpha)) \\ &= \widehat{\delta}(t,\alpha)(\epsilon) \\ &= t{\downarrow}\alpha(\epsilon) \\ &= t(\alpha)\ . \end{aligned}$$

Therefore $t = t_{\mathcal{M}}$.

(ii) $\Longrightarrow$ (i) For any term automaton $\mathcal{M}$ and α, $\beta \in \omega^*$, a straightforward inductive argument shows that

$$\widehat{\delta}(q_0,\alpha\beta) = \widehat{\delta}(\widehat{\delta}(q_0,\alpha),\beta)\ ,$$

thus

$$\begin{aligned} t_{\mathcal{M}}{\downarrow}\alpha &= \lambda\beta.t_{\mathcal{M}}(\alpha\beta) \\ &= \lambda\beta.\ell(\widehat{\delta}(q_0,\alpha\beta)) \\ &= \lambda\beta.\ell(\widehat{\delta}(\widehat{\delta}(q_0,\alpha),\beta)) \\ &= t_{\mathcal{M}_\alpha}\ , \end{aligned}$$

where $\mathcal{M}_\alpha$ is $\mathcal{M}$ with start state $\widehat{\delta}(q_0,\alpha)$ (if it exists) instead of q_0. If $\widehat{\delta}(q_0,\alpha)$ does not exist, then $t_{\mathcal{M}}{\downarrow}\alpha$ has empty domain. Thus $t_{\mathcal{M}}$ has no more subterms than there are states of $\mathcal{M}$.

The equivalence of (i) and (iii) is proved in [5]. $\square$

4 Types

Types are terms over the ranked alphabet $\Sigma = \{\bot, \rightarrow, \top\}$, where $\rightarrow$ is binary and $\bot$, $\top$ are nullary. Over this signature, every $\mathcal{D}(t) \subseteq \{0,1\}^*$. At the risk of ambiguity, we omit the superscript T_Σ on the derived operators $\rightarrow^{T_\Sigma}$, $\bot^{T_\Sigma}$, $\top^{T_\Sigma}$ and use infix notation for $\rightarrow$; thus we write $s \rightarrow t$ for the term with left subterm s and right subterm t, and $\bot$ and $\top$ for the singleton terms with the corresponding labels.

The finite types F_Σ are ordered naturally by the following inductively defined binary relation $\leq_{\mathrm{FIN}}$. This relation captures the natural type inclusion or coercion order in that it is covariant in the range and contravariant in the domain of a function type.

Definition 9 The order $\leq_{\mathrm{FIN}}$ is the smallest binary relation on F_Σ such that

(i) $\bot \leq_{\mathrm{FIN}} t \leq_{\mathrm{FIN}} \top$ for all finite t;

(ii) if $s' \leq_{\mathrm{FIN}} s$ and $t \leq_{\mathrm{FIN}} t'$ then $s \rightarrow t \leq_{\mathrm{FIN}} s' \rightarrow t'$.

□

We remark that the converse of Definition 9(ii) holds as well, since F_Σ is a free algebra.

In order to handle recursive types, we need to extend the ordering $\leq_{\mathrm{FIN}}$ to infinite types in a natural way. Much of the effort in Amadio and Cardelli's paper [1] is devoted to this task. Their definition, which involves infinite sequences of finite approximations, is given later (Definition 15). Here we give a simplified definition that does not involve approximations (Definition 11). We will eventually show (Theorem 16) that the two definitions are equivalent.

Definition 10 The *parity* of $\alpha \in \{0,1\}^*$ is the number mod 2 of 0's in α. The parity of α is denoted $\pi\alpha$. A string α is said to be *even* if $\pi\alpha = 0$ and *odd* if $\pi\alpha = 1$. □

Definition 11 Let $\leq_0$ be the linear order

$$\bot \;\leq_0\; \rightarrow \;\leq_0\; \top$$

on Σ, and let $\leq_1$ be its reverse

$$\top \quad \leq_1 \quad \rightarrow \quad \leq_1 \quad \perp .$$

For $s, t \in T_\Sigma$, define $s \leq t$ if $s(\alpha) \leq_{\pi\alpha} t(\alpha)$ for all $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$. $\square$

Lemma 12 *The relation $\leq$ is a partial order on T_Σ, and agrees with $\leq_{\mathrm{FIN}}$ on F_Σ. In particular, for any s, t, s', t',*

(i) $\perp \leq t \leq \top$

(ii) $t \leq \perp$ *if and only if* $t = \perp$

(iii) $\top \leq t$ *if and only if* $t = \top$

(iv) $s \rightarrow t \leq s' \rightarrow t'$ *if and only if* $s' \leq s$ *and* $t \leq t'$.

Proof. First we show that $\leq$ is a partial order. Reflexivity is trivial, since $\leq_{\pi\alpha}$ is a partial order.

For transitivity, suppose $s \leq t \leq u$. Let $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(u)$. Surely $\epsilon \in \mathcal{D}(t)$; and if β is a proper prefix of α in $\mathcal{D}(t)$, then

$$\rightarrow \quad = \quad s(\beta) \quad \leq_{\pi\beta} \quad t(\beta) \quad \leq_{\pi\beta} \quad u(\beta) \quad = \quad \rightarrow ,$$

so $t(\beta) = \rightarrow$, therefore β is not a leaf of $\mathcal{D}(t)$. Since $\epsilon \in \mathcal{D}(t)$ and no proper prefix of α is a leaf of $\mathcal{D}(t)$, we must have $\alpha \in \mathcal{D}(t)$. But then

$$s(\alpha) \quad \leq_{\pi\alpha} \quad t(\alpha) \quad \leq_{\pi\alpha} \quad u(\alpha) ,$$

thus $s(\alpha) \leq_{\pi\alpha} u(\alpha)$ by the transitivity of $\leq_{\pi\alpha}$. Since α was arbitrary, $s \leq u$.

For antisymmetry, assume $s \leq t \leq s$. Let $\alpha \in \mathcal{D}(s)$. Arguing as above, we must have $\alpha \in \mathcal{D}(t)$, thus $\mathcal{D}(s) \subseteq \mathcal{D}(t)$, and by symmetry, $\mathcal{D}(t) \subseteq \mathcal{D}(s)$. For any $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$, we have

$$s(\alpha) \quad \leq_{\pi\alpha} \quad t(\alpha) \quad \leq_{\pi\alpha} \quad s(\alpha) ,$$

thus $s(\alpha) = t(\alpha)$. Since s and t have the same domain and agree on the intersection of their domains, they are equal.

We next establish the properties (i)–(iv) in turn.

(i) For any t, we have $\epsilon \in \mathcal{D}(t)$ and $\perp(\epsilon) \leq_0 t(\epsilon) \leq_0 \top(\epsilon)$.

(ii), (iii) follow immediately from (i) and antisymmetry.

(iv) For *if*, suppose $s' \leq s$ and $t \leq t'$ and let $\alpha \in \mathcal{D}(s \to t) \cap \mathcal{D}(s' \to t')$. If $\alpha = \epsilon$, we have

$$(s \to t)(\epsilon) \quad = \quad (s' \to t')(\epsilon) \quad = \quad \to$$

so

$$(s \to t)(\epsilon) \quad \leq_{\pi\epsilon} \quad (s' \to t')(\epsilon) \ .$$

If $\alpha = 0\beta$, then $\beta \in \mathcal{D}(s) \cap \mathcal{D}(s')$ and

$$\begin{array}{rcl} (s' \to t')(\alpha) & = & (s' \to t')(0\beta) \\ & = & s'(\beta) \\ & \leq_{\pi\beta} & s(\beta) \\ & = & (s \to t)(0\beta) \\ & = & (s \to t)(\alpha) \ , \end{array}$$

therefore

$$(s \to t)(\alpha) \quad \leq_{\pi\alpha} \quad (s' \to t')(\alpha) \ .$$

If $\alpha = 1\beta$, then $\beta \in \mathcal{D}(t) \cap \mathcal{D}(t')$ and

$$(s \to t)(\alpha) \quad \leq_{\pi\alpha} \quad (s' \to t')(\alpha)$$

by a similar argument.

For *only if*, assume that $s \to t \leq s' \to t'$. Let $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(s')$. Then $0\alpha \in \mathcal{D}(s \to t) \cap \mathcal{D}(s' \to t')$, therefore

$$\begin{array}{rcl} s(\alpha) & = & (s \to t)(0\alpha) \\ & \leq_{\pi(0\alpha)} & (s' \to t')(0\alpha) \\ & = & s'(\alpha) \ , \end{array}$$

thus $s'(\alpha) \leq_{\pi\alpha} s(\alpha)$. Since α was arbitrary, $s' \leq s$. A similar argument shows that for arbitrary $\alpha \in \mathcal{D}(t) \cap \mathcal{D}(t')$ we have $t(\alpha) \leq_{\pi\alpha} t'(\alpha)$, thus $t \leq t'$.

Finally, we show that the orders $\leq_{\mathrm{FIN}}$ and $\leq$ agree on finite types, *i.e.*, $s \leq t$ if and only if $s \leq_{\mathrm{FIN}} t$. We proceed by induction on the structure of s

and t. If $s = \bot$ or $t = \top$ then the result follows from (i). If $t = \bot$ then the result is immediate from (ii), and if $s = \top$ then the result is immediate from (iii). The remaining case

$$s \to t \leq s' \to t' \iff s \to t \leq_{\mathrm{FIN}} s' \to t'$$

follows immediately from (iv) and the induction hypothesis on the subterms.

$\square$

In order to define Amadio and Cardelli's order, we have to consider finite approximations to infinite terms. This is done using a *truncation operator*.

Definition 13 Let $g : \omega^* \to \{\bot, \top\}$. For any term t, we define a finite term $t|_k^g$, the *level-k truncation of t with respect to g*, as follows:

$$\begin{array}{rcl} \mathcal{D}(t|_k^g) & = & \{\alpha \in \mathcal{D}(t) \mid |\alpha| \leq k\} \\ t|_k^g(\alpha) & = & \left\{ \begin{array}{ll} t(\alpha)\ , & \text{if } |\alpha| < k, \\ g(\alpha)\ , & \text{if } |\alpha| = k. \end{array} \right. \end{array}$$

$\square$

In otherwords, $t|_k^g$ is obtained by truncating the term t at depth k and relabeling the leaves according to g.

For example, let

$$\mathrm{AC}(\alpha) = \left\{ \begin{array}{ll} \bot\ , & \text{if } \alpha \text{ even}, \\ \top\ , & \text{if } \alpha \text{ odd}. \end{array} \right.$$

The truncation operator $|_k^{\mathrm{AC}}$ is the one employed by Amadio and Cardelli [1]. It has the nice property that $t|_k^{\mathrm{AC}} \leq_{\mathrm{FIN}} t|_{k+1}^{\mathrm{AC}}$, although this property turns out not to be essential. The following lemma shows that the particular function g chosen in the definition of the truncation operator is irrelevant for our purposes.

Define $s \leq^k t$ if $s(\alpha) \leq_{\pi\alpha} t(\alpha)$ for all $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$ such that $|\alpha| < k$. Note that g is not mentioned in this condition.

Lemma 14 *For any function $g : \omega^* \to \{\bot, \top\}$, terms s and t, and $k \geq 0$,*

$$s|_k^g \leq t|_k^g \quad \textit{iff} \quad s \leq^k t\ .$$

Proof. Suppose $s|_k^g \leq t|_k^g$. If $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$ and $|\alpha| < k$, then $\alpha \in \mathcal{D}(s|_k^g) \cap \mathcal{D}(t|_k^g)$ and

$$s(\alpha) \quad = \quad s|_k(\alpha) \quad \leq_{\pi\alpha} \quad t|_k(\alpha) \quad = \quad t(\alpha)\ .$$

Conversely, suppose $s \leq^k t$. If $\alpha \in \mathcal{D}(s|_k) \cap \mathcal{D}(t|_k)$ and $|\alpha| < k$, then $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$ and

$$s|_k(\alpha) \quad = \quad s(\alpha) \quad \leq_{\pi\alpha} \quad t(\alpha) \quad = \quad t|_k(\alpha)\ ,$$

and if $\alpha \in \mathcal{D}(s|_k) \cap \mathcal{D}(t|_k)$ and $|\alpha| = k$, then $s|_k(\alpha) = t|_k(\alpha) = g(\alpha)$. □

Definition 15 Amadio-Cardelli's order $\leq_{\mathrm{AC}}$ is defined as

$$s \leq_{\mathrm{AC}} t \iff s|_k^{\mathrm{AC}} \leq_{\mathrm{FIN}} t|_k^{\mathrm{AC}} \text{ for all } k \geq 0.$$

□

By Lemmas 12 and 14, this definition is independent of the choice of truncation operator.

Theorem 16 *The relations $\leq$ and $\leq_{\mathrm{AC}}$ agree.*

Proof. For any terms s and t,

$$\begin{aligned} s \leq t &\iff s \leq^k t \text{ for all } k \geq 0 \\ &\iff s|_k^{\mathrm{AC}} \leq t|_k^{\mathrm{AC}} \text{ for all } k \geq 0, \text{ by Lemma 14} \\ &\iff s|_k^{\mathrm{AC}} \leq_{\mathrm{FIN}} t|_k^{\mathrm{AC}} \text{ for all } k \geq 0, \text{ by Lemma 12} \\ &\iff s \leq_{\mathrm{AC}} t\ . \end{aligned}$$

□

5 An Algorithm

In this section we give an algorithm to decide whether $s \leq t$ for two given regular types s and t. Assume s and t are given by term automata $\mathcal{M}$ and $\mathcal{N}$ respectively over the ranked alphabet $\Sigma = \{\bot, \rightarrow, \top\}$. If s and t are given by other means, say by simultaneous equations as in [1], then results of [5] can be used to obtain the automata in linear time as described in Lemma 8 of Section 3.

Recall from Definition 11 that $s \leq t$ iff $s(\alpha) \leq_{\pi\alpha} t(\alpha)$ for all $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$. Equivalently, $s \not\leq t$ iff the set

$$\{\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t) \mid s(\alpha) \not\leq_{\pi\alpha} t(\alpha)\} \tag{1}$$

is nonempty. We show that the set (1) is a regular subset of $\{0,1\}^*$, and describe a conventional finite automaton $\mathcal{A}$ (in the sense of [6]) over the input alphabet $\{0,1\}$ that accepts exactly this set.

Define

$$\mathcal{A} \;\; = \;\; (Q^{\mathcal{A}},\ \{0,1\},\ q_0^{\mathcal{A}},\ \delta^{\mathcal{A}},\ F^{\mathcal{A}})$$

where:

- $Q^{\mathcal{A}} = Q^{\mathcal{M}} \times Q^{\mathcal{N}} \times \{0,1\}$ are the states of $\mathcal{A}$;
- $q_0^{\mathcal{A}} = (q_0^{\mathcal{M}}, q_0^{\mathcal{N}}, 0)$ is the start state of $\mathcal{A}$;
- $\delta^{\mathcal{A}} : Q^{\mathcal{A}} \times \{0,1\} \to Q^{\mathcal{A}}$ is the partial function which for $b, i \in \{0,1\}$, $p \in Q^{\mathcal{M}}$, and $q \in Q^{\mathcal{N}}$ gives

  $$\delta^{\mathcal{A}}((p,q,b),i) \;\; = \;\; (\delta^{\mathcal{M}}(p,i), \delta^{\mathcal{N}}(q,i), b \oplus \pi i)$$

  where $\oplus$ denotes mod 2 sum;
- the set of accept states of $\mathcal{A}$ is

  $$F^{\mathcal{A}} \;\; = \;\; \{(p,q,b) \mid \ell^{\mathcal{M}}(p) \not\leq_b \ell^{\mathcal{N}}(q)\}\ .$$

According to this definition, $\delta^{\mathcal{A}}((p,q,b),i)$ is defined if and only if $\ell^{\mathcal{M}}(p) = \ell^{\mathcal{N}}(q) = \rightarrow$. The automaton $\mathcal{A}$ is nondeterministic only in the sense that the state (p,q,b) has no i-successors if either $\ell^{\mathcal{M}}(p)$ or $\ell^{\mathcal{N}}(q) \in \{\bot, \top\}$. If $\ell^{\mathcal{M}}(p) = \ell^{\mathcal{N}}(q) = \rightarrow$, then the i-successor of (p,q,b) is defined and is unique.

Theorem 17 *The automaton $\mathcal{A}$ accepts the set (1).*

Proof. Extend the partial function $\delta^{\mathcal{A}}$ to a partial function

$$\widehat{\delta}^{\mathcal{A}} : Q^{\mathcal{A}} \times \{0,1\}^* \quad \to \quad Q^{\mathcal{A}}$$

inductively as usual:

$$\begin{array}{rcl} \widehat{\delta}^{\mathcal{A}}(p,\epsilon) & = & p \\ \widehat{\delta}^{\mathcal{A}}(p,\alpha i) & = & \delta^{\mathcal{A}}(\widehat{\delta}^{\mathcal{A}}(p,\alpha),i) \ . \end{array}$$

By definition, α is accepted by $\mathcal{A}$ iff $\widehat{\delta}^{\mathcal{A}}(q_0^{\mathcal{A}},\alpha)$ exists and is in $F^{\mathcal{A}}$.

We show by induction that for any $\alpha \in \{0,1\}^*$, $p \in Q^{\mathcal{M}}$, $q \in Q^{\mathcal{N}}$, and $b \in \{0,1\}$,

$$\widehat{\delta}^{\mathcal{A}}((p,q,b),\alpha) \quad = \quad (\widehat{\delta}^{\mathcal{M}}(p,\alpha),\widehat{\delta}^{\mathcal{N}}(q,\alpha),b \oplus \pi\alpha) \ . \tag{2}$$

(Of course, the use of the equality symbol $=$ to compare expressions involving partial functions bears the extra semantic condition that the left hand side is defined if and only if the right hand side is. This is an implicit but important part of our equational arguments.)

For the basis $\alpha = \epsilon$, we have

$$\begin{array}{rcl} \widehat{\delta}^{\mathcal{A}}((p,q,b),\epsilon) & = & (p,q,b) \\ & = & (\widehat{\delta}^{\mathcal{M}}(p,\epsilon),\widehat{\delta}^{\mathcal{N}}(q,\epsilon),b \oplus \pi\epsilon) \ . \end{array}$$

For the induction step,

$$\begin{array}{rcl} \widehat{\delta}^{\mathcal{A}}((p,q,b),\alpha i) & = & \delta^{\mathcal{A}}(\widehat{\delta}^{\mathcal{A}}((p,q,b),\alpha),i) \\ & = & \delta^{\mathcal{A}}((\widehat{\delta}^{\mathcal{M}}(p,\alpha),\widehat{\delta}^{\mathcal{N}}(q,\alpha),b \oplus \pi\alpha),i) \\ & = & (\delta^{\mathcal{M}}(\widehat{\delta}^{\mathcal{M}}(p,\alpha),i),\delta^{\mathcal{N}}(\widehat{\delta}^{\mathcal{N}}(q,\alpha),i),b \oplus \pi\alpha \oplus \pi i) \\ & = & (\widehat{\delta}^{\mathcal{M}}(p,\alpha i),\widehat{\delta}^{\mathcal{N}}(q,\alpha i),b \oplus \pi(\alpha i)) \ . \end{array}$$

¿From (2) we have that the domain of the partial function

$$\lambda\alpha.\widehat{\delta}^{\mathcal{A}}((p,q,b),\alpha)$$

is the intersection of the domains of $\lambda\alpha.\widehat{\delta}^{\mathcal{M}}(p,\alpha)$ and $\lambda\alpha.\widehat{\delta}^{\mathcal{N}}(q,\alpha)$. This says that any string α accepted by $\mathcal{A}$ must lie in the set

$$\begin{array}{rcl} \mathcal{D}(\lambda\alpha.\widehat{\delta}^{\mathcal{A}}(q_0^{\mathcal{A}},\alpha)) & = & \mathcal{D}(\lambda\alpha.\widehat{\delta}^{\mathcal{M}}(q_0^{\mathcal{M}},\alpha)) \cap \mathcal{D}(\lambda\alpha.\widehat{\delta}^{\mathcal{N}}(q_0^{\mathcal{N}},\alpha)) \\ & = & \mathcal{D}(s) \cap \mathcal{D}(t) \ . \end{array}$$

For such strings α,

$$\begin{aligned}
\alpha \text{ is accepted by } \mathcal{A} &\iff \widehat{\delta}^{\mathcal{A}}(q_0^{\mathcal{A}}, \alpha) \in F^{\mathcal{A}} \\
&\iff \widehat{\delta}^{\mathcal{A}}((q_0^{\mathcal{M}}, q_0^{\mathcal{N}}, 0), \alpha) \in F^{\mathcal{A}} \\
&\iff (\widehat{\delta}^{\mathcal{M}}(q_0^{\mathcal{M}}, \alpha), \widehat{\delta}^{\mathcal{N}}(q_0^{\mathcal{N}}, \alpha), 0 \oplus \pi\alpha) \in F^{\mathcal{A}} \\
&\iff (\widehat{\delta}^{\mathcal{M}}(q_0^{\mathcal{M}}, \alpha), \widehat{\delta}^{\mathcal{N}}(q_0^{\mathcal{N}}, \alpha), \pi\alpha) \in F^{\mathcal{A}} \\
&\iff \ell^{\mathcal{M}}(\widehat{\delta}^{\mathcal{M}}(q_0^{\mathcal{M}}, \alpha)) \not\leq_{\pi\alpha} \ell^{\mathcal{N}}(\widehat{\delta}^{\mathcal{N}}(q_0^{\mathcal{N}}, \alpha)) \\
&\iff s(\alpha) \not\leq_{\pi\alpha} t(\alpha)\ .
\end{aligned}$$

Thus $\mathcal{A}$ accepts the set (1). □

To decide whether $s \leq t$, we construct the automaton $\mathcal{A}$ and ask whether it accepts a nonempty set, *i.e.*, whether there exists a path from the start state to some final state. This can be determined in linear time in the size of $\mathcal{A}$ using depth first search.

The automaton $\mathcal{A}$ has $2 \cdot |Q^{\mathcal{M}}| \cdot |Q^{\mathcal{N}}|$ states and at most two transition edges from each state. Thus the entire algorithm takes no more than $O(|s| \cdot |t|)$ time, where $|s|$ and $|t|$ are the sizes of the representations of the regular terms s and t. We have shown

Theorem 18 *The subtype relation for recursive types can be decided in time $O(n^2)$.*

This result generalizes to an arbitrary signature of type constructors, each having for every argument a given polarity.

Acknowledgements

This work was supported by the Danish Research Academy, the National Science Foundation, the John Simon Guggenheim Foundation, the U.S. Army Research Office through the ACSyAM branch of the Mathematical Sciences Institute of Cornell University under contract DAAL03-91-C-0027, and the Danish Research Council under the DART Project (5.21.08.03). An earlier version of this paper appeared as [8].

References

[1] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. In *Eighteenth Symposium on Principles of Programming Languages*, pages 104–118. ACM Press, January 1991. To appear, Trans. on Prog. Lang. and Systems.

[2] Luca Cardelli. Amber. In *Combinators and Functional Programming Languages, Proc. 13th Summer School School of the LITP*. Springer-Verlag (*LNCS* 242), May 1985.

[3] Luca Cardelli. Typeful programming. In *Lect. Notes for the IFIP Advanced Seminar on Formal Methods in Programming Language Semantics*, 1989.

[4] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17:4:471–522, December 1985.

[5] Bruno Courcelle. Fundamental properties of infinite trees. *Theor. Comput. Sci.*, 25:95–169, 1983.

[6] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[7] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient inference of partial types. In *Proc. 33rd IEEE Symp. Found. Comput. Sci.*, October 1992. To appear. Also PB-394, Computer Science Department, Aarhus University, April 1992.

[8] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient recursive subtyping. In *Proc. 20th ACM Symp. Princip. Programing Lang.*, pages 419–428. ACM, January 1993.

[9] Patrick M. O'Keefe and Mitchell Wand. Type inference for partial types is decidable. In *Proc. ESOP'92, European Symposium on Programming*. Springer-Verlag (*LNCS* 582), 1992.

[10] Satish Thatte. Type inference with partial types. In *Proc. International Colloquium on Automata, Languages, and Programming 1988*. Springer-Verlag (*LNCS* 317), 1988.