Efficient Recursive Subtyping

Jens Palsberg

Computer Science Department
Aarhus University

Joint work with
Dexter Kozen, Cornell University, and
Michael Schwartzbach, Aarhus University

The Result

An efficient algorithm that decides if one recursive type is a subtype of another.

$$t ::= v \mid \bot \mid \top \mid t \to t \mid \mu v.t \qquad \text{(types)}$$

Amadio and Cardelli (POPL'91): Exponential time.

Kozen, Palsberg, and Schwartzbach (POPL'93): $O(n^2)$ time.

Outline of the talk:

1. Recursive types and subtyping
2. Amadio and Cardelli's algorithm
3. Our algorithm

Lambda Calculus

$$E ::= x \mid \lambda x.E \mid EE \qquad (\lambda\text{-terms})$$

$$t ::= v \mid \bot \mid \top \mid t \to t \mid \mu v.t \qquad (\text{types})$$

Typing rules:

$$A \vdash x : t \quad (A(x) = t)$$

$$\frac{A[x \leftarrow t_1] \vdash E : t_2}{A \vdash (\lambda x.E) : t_1 \to t_2}$$

$$\frac{A \vdash E_1 : t_1 \to t_2 \quad A \vdash E_2 : t_1}{A \vdash (E_1 E_2) : t_2}$$

$$\frac{A \vdash E : t \quad t \leq t'}{A \vdash E : t'}$$

Recursive Types

Motivation: typing recursive functions and data structures.

A recursive type $\mu v.t$ satisfies

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

Deciding type equality: $O(n\alpha(n))$ time.

Subtyping

Motivation: typing functions in object-oriented languages.

Type ordering:

$t \leq \top$ for any type t
$\perp \leq t$ for any type t

Add the usual rule for ordering function spaces:

$s \to t \leq s' \to t'$ if and only if $s' \leq s$ and $t \leq t'$

How can we order recursive types?

Recursive Types + Subtyping

Found in Amber and Quest.

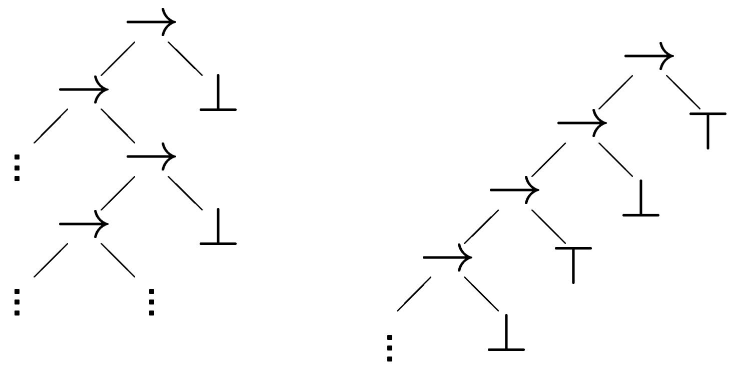
$$t ::= v \mid \bot \mid \top \mid t \to t \mid \mu v.t \qquad \text{(types)}$$

Studied by Amadio and Cardelli [POPL'91].

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

Representing types as labeled trees:

$$\mu u.((u \to u) \to \bot) \;\leq\; \mu v.((v \to \bot) \to \top)$$



Amadio and Cardelli's Algorithm

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

Definition
Two trees are ordered if all their finite truncations are ordered.

Concrete representation of trees involving back-pointers.

Eksplicit construction of a binary proof tree of polynomial depth.

Time complexity: exponential.

Our Algorithm

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

Theorem
Our definition is equivalent to Amadio and Cardelli's.

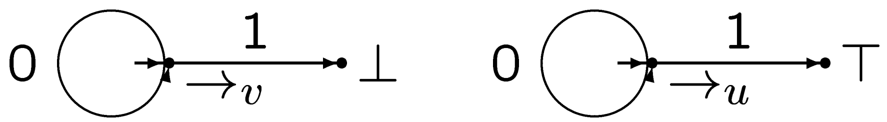
Representation of trees as *term automata*.

A certain product automaton accepts all counterexamples.

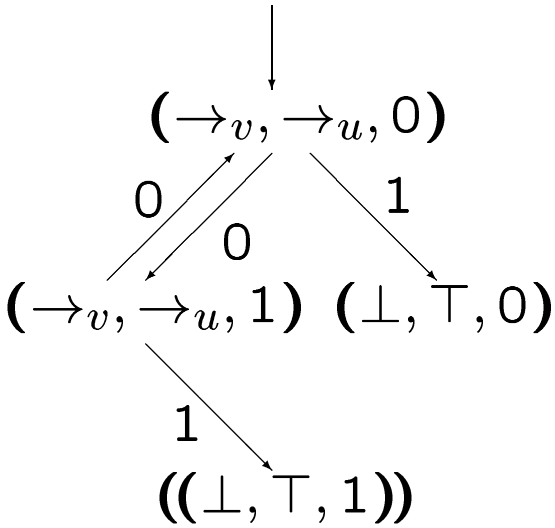
Time complexity: $O(n^2)$.

Example

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



Product automaton:



Ordering: $\bot \leq \to \leq \top$.

Perhaps reverse? 0 means "no", 1 means "yes".

Terms

Intuition: a term is a labeled tree.

Σ, a finite ranked alphabet.

In our application: $\Sigma = \{\bot, \rightarrow, \top\}$

A term over Σ is a partial function

$$t : \omega\star \ \rightarrow \ \Sigma$$

with domain $\mathcal{D}(t)$ satisfying

- $\mathcal{D}(t)$ 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 over Σ is denoted T_Σ.

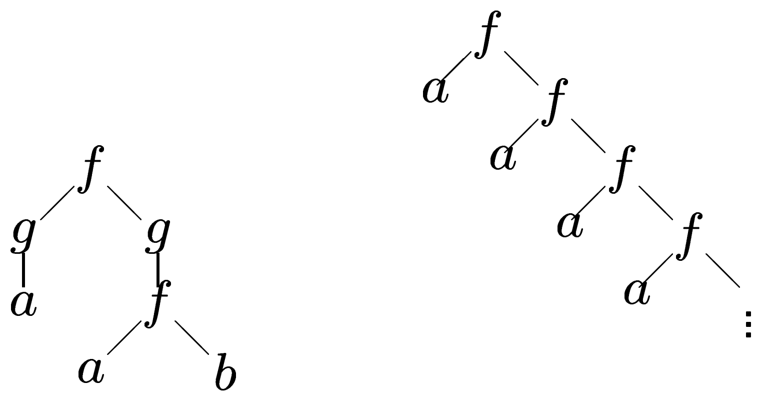
A term t is *finite* if its domain $\mathcal{D}(t)$ is a finite set.
We denote the set of finite terms over Σ by F_Σ.

Subterms

Subterms: $t{\downarrow}\alpha(\beta) = t(\alpha\beta)$

A term t is *regular* if it has only finitely many distinct subterms; *i.e.*, if $\{t{\downarrow}\alpha \mid \alpha \in \omega\star\}$ is a finite set.

Example terms:



If t is a term, then the following are equivalent:

(i) t is regular;

(ii) t is representable by a term automaton;

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

Our Type Ordering

Types are terms over $\Sigma = \{\bot, \to, \top\}$.
Over this signature, every $\mathcal{D}(t) \subseteq \{0, 1\}\star$.

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

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

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

The *parity* of $\alpha \in \{0, 1\}\star$ 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 $\pi\alpha = 1$.

Let $\leq_0$ be the linear order

$$\bot \ \leq_0 \ \to \ \leq_0 \ \top$$

on Σ, and let $\leq_1$ be its reverse

$$\top \ \leq_1 \ \to \ \leq_1 \ \bot \ .$$

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

Basic Result

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

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

(ii) $t \leq \bot$ if and only if $t = \bot$

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

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

Recall that $s \not\leq t$ iff
$\{\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t) \mid s(\alpha) \not\leq_{\pi\alpha} t(\alpha)\}$ is nonempty.

Theorem
A certain product automaton accepts

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

Algorithm

Input: Two types s and t

Output: Is $s \leq t$?

1. Construct the term automata for s and t (linear time).

2. Construct the product automaton as sketched before (size $O(n^2)$).

3. Decide, using depth first search, if the product automaton accepts the empty set (linear time).

4. If so then $s \leq t$ else $s \not\leq t$.

Time complexity: $O(n^2)$.

Tree Truncations

The *level-k truncations* $t|_k$, $t\lceil_k$, and $t\lfloor_k$ of the term t are defined by

$$\mathcal{D}(t|_k) = \mathcal{D}(t\lceil_k) = \mathcal{D}(t\lfloor_k) = \{\alpha \in \mathcal{D}(t) \mid |\alpha| \leq k\}$$

and

$$\begin{aligned}
t|_k(\alpha) &= \begin{cases} t(\alpha), & |\alpha| < k \\ \top, & |\alpha| = k \end{cases} \\
t\lceil_k(\alpha) &= \begin{cases} t(\alpha), & |\alpha| < k \\ \lceil\alpha\rceil, & |\alpha| = k \end{cases} \\
t\lfloor_k(\alpha) &= \begin{cases} t(\alpha), & |\alpha| < k \\ \lfloor\alpha\rfloor, & |\alpha| = k \end{cases}
\end{aligned}$$

where

$$\lceil\alpha\rceil \;=\; \begin{cases} \bot, & \alpha \text{ even} \\ \top, & \alpha \text{ odd,} \end{cases} \qquad \lfloor\alpha\rfloor \;=\; \begin{cases} \top, & \alpha \text{ even} \\ \bot, & \alpha \text{ odd.} \end{cases}$$

Notice that $t\lceil_k \leq_{\mathsf{FIN}} t\lceil_{k+1}$

Main Result

Lemma:
For any $k \geq 0$ and terms s and t, the following are equivalent:

- $s|_k \ \leq_{\mathsf{FIN}} \ t|_k$
- $s\lceil_k \leq_{\mathsf{FIN}} \ t\lceil_k$
- $s\lfloor_k \leq_{\mathsf{FIN}} \ t\lfloor_k$.

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

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

Theorem: The relation $\leq$ agrees with $\leq_{\mathsf{AC}}$.

Conclusion

- Automata!
- Fast algorithms for type systems based on contravariance.

Future work
Extend the algorithms to handle more constant types.