Constrained Types and their Expressiveness

Jens Palsberg[*] Scott Smith[†]

October 4, 1995

Abstract

A constrained type consists of both a standard type and a constraint set. Such types enable efficient type inference for object-oriented languages with polymorphism and subtyping, as demonstrated by Eifrig, Smith, and Trifonov. Until now, it has been unclear how expressive constrained types are.

In this paper we prove that for a language without polymorphism, constrained types accept the same programs as the type system of Amadio and Cardelli with subtyping and recursive types. This result gives a precise connection between constrained types and the standard notion of type.

1 Introduction

A constrained type consists of both a standard type and a constraint set. For example,

$$\lambda x.xx : (v \to w) \setminus \{v \leq v \to w\}$$

Here, v and w are type variables. This typing says that the λ-term $\lambda x.xx$ has every type of the form $v \to w$ where v, w satisfy the constraint $v \leq v \to w$.

[*]Jens Palsberg, Laboratory for Computer Science, Massachusetts Institute of Technology, NE43-340, 545 Technology Square, Cambridge, MA 02139; email: `palsberg@theory.lcs.mit.edu`.

[†]Scott Smith, Department of Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218; email: `scott@cs.jhu.edu`.

When combined with universal quantification, such types enable efficient type inference for object-oriented languages with polymorphism and subtyping, as demonstrated by Eifrig, Smith, and Trifonov [5, 4]. Until now, it has been unclear how expressive constrained types are.

In this paper we characterize constrained types without universal quantification, that is, types of the form $t \setminus C$ where t is a simple type and C is a constraint system. Our example language is a λ-calculus generated by the grammar:

$$E ::= x \mid \lambda x.E \mid E_1 E_2 \mid \mathsf{0} \mid \mathsf{succ}\ E\ .$$

We prove that constrained types accept the same programs as the type system of Amadio and Cardelli with subtyping and recursive types [2]. In their type system, types can be presented by the following grammar:

$$t ::= t_1 \to t_2 \mid \mathsf{Int} \mid v \mid \mu v.t \mid \top \mid \bot$$

Here, $\to$ is the function type constructor, Int is the type of integers, v is a type variable, $\mu v.t$ is a recursive type, and $\top$ and $\bot$ are the greatest and the least types, respectively. Our result thus gives a precise connection between constrained types and standard types.

To illustrate what type derivations look like in the two type systems, consider the λ-term

$$\lambda x.x(\mathsf{succ}\ x)$$

In the Amadio/Cardelli type system, we can derive that this term is typable, as follows. Define $A = \emptyset[x \leftarrow \bot]$.

$$\dfrac{\dfrac{\dfrac{A \vdash x : \bot \qquad \bot \leq \mathsf{Int} \to \top}{A \vdash x : \mathsf{Int} \to \top} \qquad \dfrac{\dfrac{A \vdash x : \bot \qquad \bot \leq \mathsf{Int}}{A \vdash x : \mathsf{Int}}}{A \vdash \mathsf{succ}\ x : \mathsf{Int}}}{A \vdash x(\mathsf{succ}\ x) : \top}}{\emptyset \vdash \lambda x.x(\mathsf{succ}\ x) : \bot \to \top}$$

In the constrained type system, we can derive that this term is typable as follows. Define $B = \emptyset[x \leftarrow v]$, where v is a type variable.

$$\dfrac{\dfrac{B \vdash_c x : v \setminus \emptyset \qquad \dfrac{\dfrac{B \vdash_c x : v \setminus \emptyset \qquad v \setminus \emptyset \trianglelefteq \mathsf{Int} \setminus \{v \leq \mathsf{Int}\}}{B \vdash_c x : \mathsf{Int} \setminus \{v \leq \mathsf{Int}\}}}{B \vdash_c \mathsf{succ}\ x : \mathsf{Int} \setminus \{v \leq \mathsf{Int}\}}}{B \vdash_c x(\mathsf{succ}\ x) : w \setminus \{v \leq \mathsf{Int},\ v \leq \mathsf{Int} \to w\}}}{\emptyset \vdash_c \lambda x.x(\mathsf{succ}\ x) : v \to w \setminus \{v \leq \mathsf{Int},\ v \leq \mathsf{Int} \to w\}}$$

One may understand the constrained type system as producing a representation of a range of possible types rather than just a single type. For instance, in this example no constraints are imposed on w in the constrained type derivation, so it may be any type and not just $\top$.

Type inference for the Amadio/Cardelli type system is computable in $O(n^3)$ time, where n is the size of the λ-term [7]. Similarly, type inference for the constrained type system is computable in $O(n^3)$ time [5]. The two systems accept the same programs as a certain flow analysis [7].

In the following two sections we recall the definitions of the Amadio/Cardelli type system and the constrained type system, and in Section 4 we prove our result.

2 The Amadio/Cardelli type system

We first define the notions of type and term. Instead of writing types in the syntax suggested above, we represent them as regular trees [2, 6]. Such trees are in turn represented by terms.

Definition 2.1 Let $\Sigma = \{\rightarrow, \mathsf{Int}, \bot, \top\}$ be the ranked alphabet where $\rightarrow$ is binary and $\mathsf{Int}, \bot, \top$ are nullary. A *type* is a regular tree over Σ. A *path* from the root of such a tree is a string over $\{0,1\}$, where 0 indicates "left subtree," and 1 indicates "right subtree." $\Box$

Definition 2.2 We represent a type by a *term*, that is, a partial function

$$t : \{0,1\}^* \rightarrow \Sigma$$

with domain $\mathcal{D}(t)$ where t maps each path from the root of the type to the symbol at the end of the path. The set of all such terms is denoted T_Σ. $\Box$

Types are ordered by the subtype relation $\leq$, as follows.

Definition 2.3 The *parity* of $\alpha \in \{0,1\}^*$ is the number mod 2 of zeros 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 partial order on Σ given by

$$\begin{array}{lll} \bot \leq_0 \rightarrow & \text{and} & \rightarrow \leq_0 \top \text{ and} \\ \bot \leq_0 \mathsf{Int} & \text{and} & \mathsf{Int} \leq_0 \top\ . \end{array}$$

Let $\leq_1$ be its reverse

$$\begin{array}{lcl} \top \leq_1 \to & \text{and} & \to \leq_1 \bot \text{ and} \\ \top \leq_1 \mathsf{Int} & \text{and} & \mathsf{Int} \leq_1 \bot \;. \end{array}$$

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

Kozen et al. [6] showed that the relation $\leq$ is equivalent to the order defined by Amadio and Cardelli [2]. The relation $\leq$ is a partial order, and if $s \to t \leq s' \to t'$, then $s' \leq s$ and $t \leq t'$ [2, 6].

Next, we present the type rules. If E is a λ-term, t is a type, and A is a type environment, i.e., a partial function assigning types to variables, then the judgment $A \vdash E : t$ means that E has the type t in the environment A. Formally, this holds when the judgment is derivable using the following six rules:

$$A \vdash \mathsf{0} : \mathsf{Int} \tag{1}$$

$$\frac{A \vdash E : \mathsf{Int}}{A \vdash \mathsf{succ}\ E : \mathsf{Int}} \tag{2}$$

$$A \vdash x : t \quad \text{(provided } A(x) = t\text{)} \tag{3}$$

$$\frac{A[x \leftarrow s] \vdash E : t}{A \vdash \lambda x.E : s \to t} \tag{4}$$

$$\frac{A \vdash E : s \to t \quad A \vdash F : s}{A \vdash EF : t} \tag{5}$$

$$\frac{A \vdash E : s \quad s \leq t}{A \vdash E : t} \tag{6}$$

The first five rules are the usual rules for simple types, and the last rule is the rule of *subsumption*.

The type system has the subject reduction property, that is, if $A \vdash E : t$ is derivable, and E β-reduces to E', then $A \vdash E' : t$ is derivable. This is proved by straightforward induction on the structure of the derivation of $A \vdash E : t$.

3 Constrained Types

We begin with defining what will be called a simple type. The set of simple types is generated by the following grammar:

$$t ::= t_1 \to t_2 \mid \mathsf{Int} \mid v$$

Here, v is a type variable.

Definition 3.1 A *constraint system* is a finite set of constraints of the form $t \leq t'$, where t, t' are simple types.

A constraint system C is *closed* if the following two conditions hold.

- If $s \rightarrow t \leq s' \rightarrow t'$ is in C, then $s' \leq s$ and $t \leq t'$ are in C.
- If $r \leq s$ and $s \leq t$ both are in C, then $r \leq t$ is in C.

If C is a constraint system, then the *closure* of C is the smallest closed constraint system which contains C. If C, C' are two constraint systems, we denote by $C \uplus C'$ the closure of $C \cup C'$.

A constraint system is *consistent* if it does not contain constraints of the forms $\mathsf{Int} \leq t \rightarrow t'$ or $t \rightarrow t' \leq \mathsf{Int}$, where t, t' are simple types. $\Box$

A constrained type is of the form $t \setminus C$ where t is a simple type and C is a closed constraint system. Constrained types are ordered by the subtype relation $\trianglelefteq$, as follows.

Definition 3.2 For constrained types $t \setminus C$ and $t' \setminus C'$, define $t \setminus C \trianglelefteq t' \setminus C'$ if either $C \uplus \{t \leq t'\} \subseteq C'$, or $t = t'$ and $C \subseteq C'$. $\Box$

Notice that $\trianglelefteq$ is a partial order.

Next, we present the type rules. If E is a λ-term, $t \setminus C$ is a constrained type, and A is a simple type environment, i.e., a partial function assigning simple types to variables, then the judgment $A \vdash_c E : t \setminus C$ holds when it is derivable using the following six rules:

$$A \vdash_c \mathsf{0} : \mathsf{Int} \setminus \emptyset \tag{7}$$

$$\frac{A \vdash_c E : \mathsf{Int} \setminus C}{A \vdash_c \mathsf{succ}\ E : \mathsf{Int} \setminus C} \tag{8}$$

$$A \vdash_c x : t \setminus \emptyset \quad \text{(provided } A(x) = t\text{)} \tag{9}$$

$$\frac{A[x \leftarrow s] \vdash_c E : t \setminus C}{A \vdash_c \lambda x.E : s \rightarrow t \setminus C} \tag{10}$$

$$\frac{A \vdash_c E : s \rightarrow t \setminus C_1 \quad A \vdash_c F : s \setminus C_2}{A \vdash_c EF : t \setminus C_1 \uplus C_2} \tag{11}$$

$$\frac{A \vdash_c E : t \setminus C \quad t \setminus C \trianglelefteq t' \setminus C'}{A \vdash_c E : t' \setminus C'} \tag{12}$$

Notice that there is a rule for each syntactic construct and also a subsumption rule. It is the subsumption rule that makes it possible to add constraints to the constraint set. If $A \vdash_c E : t \setminus C$ is derivable and C is consistent, then we say that E has the constrained type $t \setminus C$ in the environment A. Notice that the existence of a derivation of $A \vdash_c E : t \setminus C$ does not imply that E is typable, since C need not be consistent. In the proof of Lemma 4.2 below we will prove that certain derivations exist without considering the issue of consistency.

Soundness of a more general set of rules than (7)–(12) is established in [5] by subject reduction, which establishes that if $A \vdash_c E : t \setminus C$ and C is consistent, execution of E will not result in type errors. Other forms of constrained type are presented in [1, 3].

4 Equivalence

We now establish that the Amadio/Cardelli type system and the constrained type system are equivalent in power. To prove the result independently would be a significant effort, but using facts already proven in [7] and [5], it is not difficult.

Given a λ-term E, we now describe how to generate a certain constraint system, found in [7]. Assume that E has been α-converted so that all bound variables are distinct. Let X_E be a set of type variables consisting of one type variable $\langle x \rangle$ for each λ-variable x occurring in E, and let Y_E be a set of variables disjoint from X_E consisting of one variable $[\![F]\!]$ for each occurrence of a subterm F of E. (The notation $[\![F]\!]$ is ambiguous because there may be more than one occurrence of F in E. However, it will always be clear from context which occurrence is meant.) The following constraint system uses $X_E \cup Y_E$ as type variables.

- for every occurrence in E of a subterm of the form $\mathsf{0}$, the inequality

$$\mathsf{Int} \;\leq\; [\![\mathsf{0}]\!] \;;$$

- for every occurrence in E of a subterm of the form $\mathsf{succ}\ F$, the two inequalities

$$\begin{aligned} \mathsf{Int} &\leq [\![\mathsf{succ}\ F]\!] \\ [\![F]\!] &\leq \mathsf{Int}\ ; \end{aligned}$$

- for every occurrence in E of a subterm of the form $\lambda x.F$, the inequality

$$\langle x \rangle \to [\![F]\!] \ \leq \ [\![\lambda x.F]\!]\ ;$$

- for every occurrence in E of a subterm of the form GH, the inequality

$$[\![G]\!] \ \leq \ [\![H]\!] \to [\![GH]\!]\ ;$$

- for every occurrence in E of a λ-variable x, the inequality

$$\langle x \rangle \ \leq \ [\![x]\!]\ .$$

Denote by $T(E)$ the system of constraints generated from E in this fashion. The closure of $T(E)$ will be written $\overline{T}(E)$.

If C is a constraint system and φ is a function that maps the type variables used in C to elements of T_Σ such that all constraints are satisfied, then φ is a *solution* of C. We say that C is *solvable* if it has a solution.

Theorem 4.1 *For a λ-term E, the following two conditions are equivalent:*

1. *E is typable in the Amadio/Cardelli type system.*
2. *$\overline{T}(E)$ is consistent.*

Proof. In [7] there is a notion of closure which we here will call restricted closure. It is defined as follows. A constraint system C is *restricted-closed* if the following two conditions hold.

- If $s \to t \leq s' \to t'$ is in C, then $s' \leq s$ and $t \leq t'$ are in C.
- If $r \leq v$ and $v \leq t$ both are in C, then $r \leq t$ is in C.

Here, v is a type variable. If C is a constraint system, then the *restricted closure* of C is the smallest restricted-closed constraint system which contains C. The restriction is to close only under transitivity through variables. The different definitions of closure are solely an artifact of slight differences of approach in the two papers [5, 7].

We will prove that the following five properties are equivalent:

1. E is typable in the Amadio/Cardelli type system.

2. The restricted closure of $T(E)$ is solvable.

3. The restricted closure of $T(E)$ is consistent.

4. $\overline{T}(E)$ is solvable.

5. $\overline{T}(E)$ is consistent.

In [7] it is proved that (1), (2), and (3) are equivalent.

To prove (2) $\Rightarrow$ (4), suppose the restricted closure of $T(E)$ has solution φ. Then also $T(E)$ has solution φ, and since the rules that define the closure of a constraint system preserve solutions, $\overline{T}(E)$ has solution φ.

It is immediate that (4) $\Rightarrow$ (5), since no inconsistent constraint set can be solvable. Finally, since the restricted closure of $T(E)$ is a subset of $\overline{T}(E)$, we have that (5) $\Rightarrow$ (3). $\Box$

For a λ-term E, let A_E be the simple type environment which maps each λ-variable x occurring in E to $\langle x \rangle$.

Lemma 4.2 *For a λ-term E, we can derive $A_E \vdash_c E : [\![E]\!] \setminus \overline{T}(E)$.*

Proof. We can prove the following stronger property. For a λ-term E, we can for every subterm F of E derive $A_E \vdash_c F : [\![F]\!] \setminus \overline{T}(E)$. This is proved by induction on the structure of F.

In the base case, consider first $F = \mathsf{0}$. We have that $A_E \vdash_c \mathsf{0} : \mathsf{Int} \setminus \emptyset$ is derivable. Moreover, the constraint $\mathsf{Int} \leq [\![\mathsf{0}]\!]$ is in $\overline{T}(E)$. Thus, $\mathsf{Int} \setminus \emptyset \trianglelefteq [\![\mathsf{0}]\!] \setminus \overline{T}(E)$, so $A_E \vdash_c \mathsf{0} : [\![\mathsf{0}]\!] \setminus \overline{T}(E)$ is derivable. Consider then $F = x$. We have that $A_E \vdash_c x : \langle x \rangle \setminus \emptyset$ is derivable. Moreover, the constraint $\langle x \rangle \leq [\![x]\!]$ is in $\overline{T}(E)$. Thus $\langle x \rangle \setminus \emptyset \trianglelefteq [\![x]\!] \setminus \overline{T}(E)$, so $A_E \vdash_c x : [\![x]\!] \setminus \overline{T}(E)$ is derivable.

In the induction step, consider first $F = \mathsf{succ}\ G$. By the induction hypothesis, we have that $A_E \vdash_c G : [\![G]\!] \setminus \overline{T}(E)$ is derivable. Moreover, the constraint $[\![G]\!] \leq \mathsf{Int}$ is in $\overline{T}(E)$. Thus, $[\![G]\!] \setminus \overline{T}(E) \trianglelefteq \mathsf{Int} \setminus \overline{T}(E)$, so $A_E \vdash_c G : \mathsf{Int} \setminus \overline{T}(E)$ is derivable. From rule (8) we get that $A_E \vdash_c \mathsf{succ}\ G : \mathsf{Int} \setminus \overline{T}(E)$ is derivable. Also the constraint $\mathsf{Int} \leq [\![\mathsf{succ}\ G]\!]$ is in $\overline{T}(E)$. Thus, $\mathsf{Int} \setminus \overline{T}(E) \trianglelefteq [\![\mathsf{succ}\ G]\!] \setminus \overline{T}(E)$, so $A_E \vdash_c \mathsf{succ}\ G : [\![\mathsf{succ}\ G]\!] \setminus \overline{T}(E)$ is derivable.

Consider then $F = \lambda x.G$. By the induction hypothesis, we have that $A_E \vdash_c G : [\![G]\!] \setminus \overline{T}(E)$ is derivable. Noting that $A_E = A_E[x \leftarrow \langle x \rangle]$, we get from rule (10) that $A_E \vdash_c \lambda x.G : \langle x \rangle \to [\![G]\!] \setminus \overline{T}(E)$ is derivable. Moreover, the constraint $\langle x \rangle \to [\![G]\!] \leq [\![\lambda x.G]\!]$ is in $\overline{T}(E)$. Thus, $\langle x \rangle \to [\![G]\!] \setminus \overline{T}(E) \trianglelefteq [\![\lambda x.G]\!] \setminus \overline{T}(E)$, so $A_E \vdash_c \lambda x.G : [\![\lambda x.G]\!] \setminus \overline{T}(E)$ is derivable.

Finally, consider $F = GH$. By the induction hypothesis, we have that both $A_E \vdash_c G : [\![G]\!] \setminus \overline{T}(E)$ and $A_E \vdash_c H : [\![H]\!] \setminus \overline{T}(E)$ are derivable. Moreover, the constraint $[\![G]\!] \leq [\![H]\!] \to [\![GH]\!]$ is in $\overline{T}(E)$. Thus, $[\![G]\!] \setminus \overline{T}(E) \trianglelefteq [\![H]\!] \to [\![GH]\!] \setminus \overline{T}(E)$, so $A_E \vdash G : [\![H]\!] \to [\![GH]\!] \setminus \overline{T}(E)$ is derivable. From rule (11) we get that $A_E \vdash_c GH : [\![GH]\!] \setminus \overline{T}(E)$ is derivable.
□

Theorem 4.3 *For a λ-term E, if $\overline{T}(E)$ is consistent, then E is typable in the constrained type system.*

Proof. Immediate from Lemma 4.2. □

Together, Theorem 4.1 and 4.3 show that if E is typable in the Amadio/Cardelli type system, then it is also typable in the constrained type system.

To prove the converse, we first present a new set of type rules, taken from [5]. If E is a λ-term, $t \setminus C$ is a constrained type where C is a constraint set, and A is a simple type environment, i.e., a partial function assigning simple types to variables, then the judgment $A \vdash_i E : t \setminus C$ holds when it is derivable using the following five rules:

$$A \vdash_i \mathsf{0} : \mathsf{Int} \setminus \emptyset \tag{13}$$

$$\frac{A \vdash_i E : t \setminus C}{A \vdash_i \mathsf{succ}\ E : \mathsf{Int} \setminus C \uplus \{t \leq \mathsf{Int}\}} \tag{14}$$

$$A \vdash_i x : t \setminus \emptyset \quad (\text{provided } A(x) = t) \tag{15}$$

$$\frac{A[x \leftarrow v] \vdash_i E : t \setminus C}{A \vdash_i \lambda x.E : v \to t \setminus C} \tag{16}$$

$$\frac{A \vdash_i E : t_1 \setminus C_1 \quad A \vdash_i F : t_2 \setminus C_2}{A \vdash_i EF : v \setminus C_1 \uplus C_2 \uplus \{t_1 \leq t_2 \to v\}} \tag{17}$$

In both rule (16) and rule (17), v is a type variable. Notice that there is a rule for each syntactic construct but no subsumption rule. It is clear by inspection that there is at most one typing derivation for each term E modulo names chosen for fresh variables. This set of rules thus serves to define an inference algorithm, which we may show is complete.

Theorem 4.4 *If $A \vdash_c E : t \setminus C$ is derivable and C is consistent, then there exists a constrained type $t' \setminus C'$ where C' is consistent, such that $A \vdash_i E : t' \setminus C'$ is derivable.*

Proof. See [5]. $\square$

Theorem 4.5 *If $A \vdash_i E : t \setminus C$ is derivable and C is consistent, then $\overline{T}(E)$ is consistent.*

Proof. Consider a derivation of $A \vdash_i E : t \setminus C$, where C is consistent. Let $X = Y$ denote the two constraints $X \leq Y$ and $Y \leq X$. Define the constraint system D as follows.

- For every occurrence of a subterm F of E, find the unique judgment in the derivation of $A \vdash_i E : t \setminus C$ which involves F, and let that judgment be of the form $A' \vdash_i F : t' \setminus C'$. Add the constraint $[\![F]\!] = t'$ to D.

- For every occurrence of a subterm GH in E, find the associated judgment in the derivation of $A \vdash_i E : t \setminus C$ of the form $A' \vdash_i GH : v \setminus C' \uplus \{t_1 \leq t_2 \to v\}$, where v is a type variable. Add the constraint $[\![G]\!] \leq [\![H]\!] \to [\![GH]\!]$ to D.

- For every λ-variable x occurring in E, find the judgment in the derivation of $A \vdash_i E : t \setminus C$ which involves the abstraction which binds x, and let that judgment be of the form $A' \vdash_i \lambda x.F : v \to t \setminus C'$, where v is a type variable. Add the constraints $\langle x \rangle = v$ and $\langle x \rangle \to [\![F]\!] \leq [\![\lambda x.F]\!]$ to D.

Notice that $C \uplus D$ is consistent: the first operation above clearly preserves consistency. The second operation also preserves consistency, as the new constraints in the closure will always mirror existing constraints, with $[\![G]\!]$ replacing t_1, $[\![H]\!]$ replacing t_2, and $[\![H]\!]$ replacing v. The third operation preserves consistency by a similar argument.

Thus, since $\overline{T}(E)$ is clearly a subset of $C \uplus D$, $\overline{T}(E)$ is consistent. $\square$

Together, Theorem 4.1, 4.4, and 4.5 show that if E is typable in the constrained type system, then it is also typable in the Amadio/Cardelli type system.

In summary, we have proved our result.

Corollary 4.6 *A λ-term E is typable in the Amadio/Cardelli type system if and only if it is typable in the constrained type system.*

5 Conclusion

The Amadio/Cardelli type system [2], a certain kind of flow analysis [7], and a simple constrained type system [5] accept the same programs, unifying three different views of typing.

Acknowledgments We thank Trevor Jim for helpful comments on a draft of the paper. The first author was supported by BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation). The second author was partially supported by NSF grants CCR-9301340 and CCR-9312433, and ONR grant N00014-95-1-0999.

References

[1] Alexander Aiken and Edward Wimmers. Type inclusion constraints and type inference. In *Proc. Conference on Functional Programming Languages and Computer Architecture*, pages 31–41, 1993.

[2] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993. Also in Proc. POPL'91.

[3] Pavel Curtis. Constrained quantification in polymorphic type analysis. Technical Report CSL-90-1, XEROX Palo Alto Research Center, 1990.

[4] J. Eifrig, S. Smith, and V. Trifonov. Sound polymorphic type inference for objects. In *Proc. OOPSLA'95, ACM SIGPLAN Tenth Annual Conference on Object-Oriented Programming Systems, Languages and Applications*, 1995.

[5] J. Eifrig, S. Smith, and V. Trifonov. Type inference for recursively constrained types and it application to OOP. In *Proc. Mathematical Foundations of Programming Semantics*, 1995. To appear.

[6] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient recursive subtyping. *Mathematical Structures in Computer Science*, 5(1):113–125, 1995. Also in Proc. POPL'93, Twentieth Annual SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pages 419–428, Charleston, South Carolina, January 1993.

[7] Jens Palsberg and Patrick M. O'Keefe. A type system equivalent to flow analysis. *ACM Transactions on Programming Languages and Systems*, 17(4):576–599, July 1995. Also in Proc. POPL'95, 22nd Annual SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pages 367–378, San Francisco, California, January 1995.