1996-01

Lower Bounds on Type Checking Overloading

Volpano, Dennis

Information Processing Letters, 57(1), pp. 9-14, Jan 1996.
http://hdl.handle.net/10945/35272

Lower Bounds on Type Checking Overloading

Dennis M. Volpano
Department of Computer Science
Naval Postgraduate School
Monterey, CA, 93943, USA

Abstract

Smith has proposed an elegant extension of the *ML* type system for polymorphic functional languages with overloading. Type inference in his system requires solving a satisfiability problem that is undecidable if no restrictions are imposed on overloading. This short note explores the effect of recursion and the structure of type assumptions in overloadings on the problem's complexity.

Keywords: Type theory, overloading, computational complexity.

1 Introduction

The *ML* type system [Mil78, DaM82] has been studied extensively and has some well-known limitations. One practical limitation is that it prohibits overloading by allowing no more than one assumption per identifier in a type context. In an attempt to overcome this limitation, Smith gives an elegant type system that merges *ML*-style polymorphism and overloading [Smi91, Smi93, Smi94]. The system has the usual set of unquantified types given by

$$\tau ::= \alpha \mid \tau \rightarrow \tau' \mid \chi(\tau_1, \ldots, \tau_n)$$

where χ is an n-ary type constructor, like *int* or *matrix*, and α is a type variable.

The set of quantified types, or *type schemes*, is given by

$$\sigma ::= \forall \alpha_1, \ldots, \alpha_n \textbf{ with } x_1 : \tau_1, \ldots, x_m : \tau_m \,.\, \tau$$

The set $\{\alpha_1, \ldots, \alpha_n\}$ is the set of *quantified variables* of σ, $\{x_1 : \tau_1, \ldots, x_m : \tau_m\}$ the set of *constraints* of σ, and τ the *body* of σ. If there are no constraints, the **with** portion of the type scheme is omitted.

The type inference rules are given in Figure 1. We can prove *typing judgements* of the form $A \vdash e : \sigma$ where A is a finite set of assumptions of the form $x : \sigma$, called a *type context*. A type context A may contain more than one typing for an identifier x; in this case we say that x is *overloaded* in A. We use the notation $A \vdash C$ to represent $A \vdash x : \tau$, for all $x : \tau$ in C, and let $|A|$ denote the number of assumptions in type context A.

[1] Appeared in Information Processing Letters, 57(1), pp.9–14, Jan 1996. This material is based upon activities supported by the National Science Foundation under Agreement No. CCR-9400592. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 JAN 1996	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Lower Bounds on Type Checking Overloading	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Computer Science Naval Postgraduate School Monterey, CA 93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**8**	

(hypoth) $A \vdash x : \sigma$, if $x : \sigma \in A$

(→-intro)
$$\frac{A \cup \{x : \tau\} \vdash e : \tau'}{A \vdash \lambda x.\, e : \tau \to \tau'} \quad (x \text{ does not occur in } A)$$

(→-elim)
$$\frac{A \vdash e : \tau \to \tau' \qquad A \vdash e' : \tau}{A \vdash e\ e' : \tau'}$$

(let)
$$\frac{A \vdash e : \sigma \qquad A \cup \{x : \sigma\} \vdash e' : \tau}{A \vdash \textbf{let } x = e \textbf{ in } e' : \tau} \quad (x \text{ does not occur in } A)$$

(∀-intro)
$$\frac{A \cup C \vdash e : \tau \qquad A \vdash C[\bar{\alpha} := \bar{\pi}]}{A \vdash e : \forall \bar{\alpha} \textbf{ with } C \,.\, \tau} \quad (\bar{\alpha} \text{ not free in } A)$$

(∀-elim)
$$\frac{A \vdash e : \forall \bar{\alpha} \textbf{ with } C \,.\, \tau \qquad A \vdash C[\bar{\alpha} := \bar{\pi}]}{A \vdash e : \tau[\bar{\alpha} := \bar{\pi}]}$$

($\equiv_\alpha$)
$$\frac{A \vdash e : \sigma \qquad \sigma \equiv_\alpha \sigma'}{A \vdash e : \sigma'}$$

Figure 1: Typing Rules with Overloading

$$
\begin{array}{l}
+ : \mathit{real} \rightarrow \mathit{real} \rightarrow \mathit{real} \\
+ : \forall\alpha \ \mathbf{with} + : \alpha \rightarrow \alpha \rightarrow \alpha \,. \\
\qquad \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha) \\
* : \mathit{int} \rightarrow \mathit{int} \rightarrow \mathit{int} \\
* : \mathit{real} \rightarrow \mathit{real} \rightarrow \mathit{real} \\
* : \forall\alpha \ \mathbf{with} + : \alpha \rightarrow \alpha \rightarrow \alpha, * : \alpha \rightarrow \alpha \rightarrow \alpha \,. \\
\qquad \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha)
\end{array}
$$

Figure 2: A recursive type context

Observe that when C is empty the ($\forall$-intro) and ($\forall$-elim) rules reduce respectively to type generalization and specialization in the type system of Damas and Milner [DaM82]. The second hypothesis of the ($\forall$-intro) rule says that a constraint set can be moved into a type scheme only if the constraint set is satisfiable; the typing $A \vdash e : \forall\bar{\alpha}$ **with** $C \,.\, \tau$ cannot be derived unless we know that there is some way of instantiating the $\bar{\alpha}$ so that C is satisfied. This leads to the following problem:

Definition 1.1 *CS-SAT is the problem of given a constraint set C and a type context A, is there a substitution S such that $A \vdash CS$?*

In practice, we would expect a type inference algorithm to generate C by specializing the constraints of type schemes in A. So from now on, we assume that constraints in C can be formed by specializing constraints in A.

CS-SAT is undecidable as was shown by Smith [Smi91]. Type schemes in this system permit very expressive type contexts to be constructed, some of which may be recursive in the following sense.

Definition 1.2 *Let $S = \{x, y \ldots\}$ be the set of identifiers with assumptions in a type context A. Then the dependency relation of A is a binary relation R on S such that $x \,R\, y$ iff $x : \sigma \in A$ and $y : \tau$ is a constraint of σ (x and y are not necessarily distinct).*

Definition 1.3 *A type context A with dependency relation R is recursive iff $\exists x.\, x \,R^{+}\, x$.*

For example, the type context in Figure 2 is recursive due to the assumptions for $*$ and $+$. As a result of recursion, we have infinitely-many types at which $*$ and $+$ have instances.

2 Lower Bounds

Although *CS-SAT* is decidable when recursion is prohibited, it remains hard if the structure of type assumptions is not restricted, requiring nondeterministic exponential time.

Theorem 2.1 *CS-SAT is NEXPTIME complete for nonrecursive type contexts.*

Proof. Given a nonrecursive type context A and a constraint set C, construct a set E of equations as follows. For each constraint $x : \tau \in C$, nondeterministically choose an assumption $x : \forall\bar{\alpha}$ **with** $C' \,.\, \tau'$ in A, add to E equation $\tau = \tau'[\bar{\alpha} := \bar{\gamma}]$, where $\bar{\gamma}$ is new, and add the members of $C'[\bar{\alpha} := \bar{\gamma}]$ to C. Then E is unifiable iff the original set C is

satisfiable with respect to A. The size of E is at most exponential in $|A|$. Whether E is unifiable can be decided in linear time so *CS-SAT* is in NEXPTIME. For hardness, let

$$B = \{c_{p(n)+1} : \epsilon \rightarrow q_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n \rightarrow \epsilon\}$$

in the PTIME reduction of Theorem 4.2 in [VoS91]; $a_1 a_2 \cdots a_n$ is an input string x to a nondeterministic Turing machine M of exponential time complexity. Then B is satisfiable under A_x iff M accepts x. □

Next we consider the complexity of *CS-SAT* with the style of overloading permitted in the lazy functional programming language Haskell [Has92]. An identifier may be overloaded in Haskell, but only through multiple instance declarations, which give rise to what we call a *Haskell type context*. A Haskell type context is very structured.

Definition 2.1 Suppose X is the set of identifiers of a nonempty type context H. Then H is a Haskell type context if for each $x \in X$, there is a type τ_x with exactly one free type variable α_x, possibly occurring more than once, such that all assumptions for x in H are described by the set

$$\left\{ \begin{array}{c} x : \forall \bar{\gamma}_1 \ \textbf{with} \ C_1 \,.\, \tau_x[\alpha_x := \chi_1(\bar{\gamma}_1)] \\ \vdots \\ x : \forall \bar{\gamma}_n \ \textbf{with} \ C_n \,.\, \tau_x[\alpha_x := \chi_n(\bar{\gamma}_n)] \end{array} \right\}$$

where $n \geq 1$, $\chi_i \neq \chi_j$ for $i \neq j$, and

$$y : \rho \in C_i \text{ implies } y \in X,\ \rho = \tau_y[\alpha_y := \gamma], \text{ and } \gamma \in \bar{\gamma}_i.$$

If x is overloaded ($n > 1$) then the type τ_x used in forming the assumptions for x is the *least common generalization* [Rey70] of the types

$$\tau_x[\alpha_x := \chi_1(\bar{\gamma}_1)], \ldots, \tau_x[\alpha_x := \chi_n(\bar{\gamma}_n)]$$

If x has only one assumption ($n = 1$), then it is, technically speaking, not overloaded, nevertheless it may appear in a constraint. The body of a type scheme in an assumption must be formed by specializing a type with one level of type structure, or in other words, by a single type constructor χ, parameterized perhaps by one or more type variables. Further, in a constraint $y : \rho$, it must be possible to form ρ by merely renaming the free type variable of τ_y, the type used in forming all assumptions for y.

The recursive type context of Figure 2 is an example of a Haskell type context. Consider the assumptions for $+$. The bodies of its type schemes are formed by instantiating α of the type $\alpha \rightarrow \alpha \rightarrow \alpha$ with *real* and $matrix(\alpha)$, so $\tau_+ = \alpha \rightarrow \alpha \rightarrow \alpha$.

Regular tree languages recognized by DR tree automata characterize the types of identifiers in Haskell type contexts. Formally, a *k-ary*, Σ-valued tree is a mapping $t : dom(t) \rightarrow \Sigma$ where $dom(t) \subseteq \{1, \ldots, k\}^*$ is a nonempty set and closed under prefixes. We can assume for our purposes that Σ is a ranked alphabet of type constructors $\chi_1, \ldots, \chi_n$. We let F_Σ denote the set of all finite Σ-trees. A deterministic root-to-frontier (DR) Σ-tree automaton M is a pair $(\mathcal{A}, S)$ such that

1. $\mathcal{A}$ is a finite DR Σ-algebra (A, Σ), and

2. $S \in A$ is the *initial state*.

A DR Σ-algebra is a pair (A, Σ) where A is a nonempty set of states and, for every $\sigma \in \Sigma_m$ with $m > 0$ there is a partial transition function $\sigma^{\mathcal{A}} : A \rightarrow A^m$. If $\sigma \in \Sigma_0$, then $\sigma^{\mathcal{A}} \subseteq A$ [GeS84]. A *run* of M on a tree t is a mapping $g : dom(t) \rightarrow A$ such that $g(\epsilon) = S$, and if $t(w) = \sigma$, $\sigma \in \Sigma_m$ for $m > 0$, and $g(w) = a$, then

$$\sigma^{\mathcal{A}}(a) = (g(w1), g(w2), \ldots, g(wm)).$$

A run g of M on a tree t is *accepting* if $g(w) \in t(w)^{\mathcal{A}}$ for every node w in the frontier of t. The set of all trees on which M has accepting runs is the *language of* M, written $T(M)$.

Deciding whether the intersection of a sequence of DR tree automata is nonempty is EXPTIME complete. Hardness can be proved by a log-space generic reduction from polynomial-space bounded alternating Turing machines (ATMs) [Sei94]. The lower bound on *CS-SAT* for Haskell type contexts follows directly from this result while the upper bound follows from the following lemma:

Lemma 2.2 *Suppose H is a Haskell type context. If x is an identifier of H then a DR tree automaton M_x can be constructed such that*

$$T(M_x) = \{\pi \mid H \vdash x : \tau_x[\alpha_x := \pi]\}.$$

Further, such automata can be constructed for all identifiers of H in time exponential (space polynomial) in $|H|$.

Proof. Given a Haskell type context H, let X be the set of identifiers of H. Let $A = \wp(X)$ and Σ be the ranked alphabet of type constructors in H. Define $M_x = (\mathcal{A}, \{x\})$, where $\mathcal{A} = (A, \Sigma)$, and $\mathcal{A}$ is initially constructed so that $T(\mathcal{A}, \{\,\}) = F_\Sigma$. Then extend $\mathcal{A}$ with the following transitions. For every $r \subseteq X$, let $r \in \chi^{\mathcal{A}}$ iff all identifiers in r have an instance assumption in H at nullary type constructor χ. Let $\chi^{\mathcal{A}}(r) = (r_1, \ldots, r_k)$ iff all have an instance at type constructor $\chi(\alpha_1, \ldots, \alpha_k)$, for $k > 0$. Set r_j, for $1 \leq j \leq k$, contains an identifier, say z, iff there is an identifier in r whose instance assumption at χ has a constraint on z involving α_j. There are $2^{|X|}$ subsets to consider, each of which can be stored in at most $|X|$ space. □

The following theorem was proved by Seidl [Sei94] in the framework of Nipkow and Prehofer's type system [NiPr93]. A proof is given here for Smith's system.

Theorem 2.3 *CS-SAT is EXPTIME complete for Haskell type contexts.*

Proof. Let H be a Haskell type context and C a constraint set. First construct a DR tree automaton $(\mathcal{A}, \{x\})$ for each identifier x of H. By Lemma 2.2, this can be done in exponential time. Suppose that for each constraint $x : \rho \in C$, $\rho = \tau_x[\alpha_x := \pi]$, for some type π. Let g be a run of $(\mathcal{A}, \{x\})$ on π. If there is a node $w \in dom(\pi)$ such that $\pi(w) = \chi_m$, $m \geq 0$, and $g(w)$ is undefined, then reject. If π has type variables, then associate a DR tree automaton with every occurrence of a variable in π as follows. For every node w such that $\pi(w) = \beta$ and $g(w) = a$, assign to this occurrence of type variable β, tree automaton $(\mathcal{A}, a)$.

Now for every type variable α in C, with occurrences $\alpha_1, \ldots, \alpha_m$, decide whether the languages of their assigned tree automata

$$(\mathcal{A}, a_1), \ldots, (\mathcal{A}, a_m)$$

intersect by checking the emptiness of automaton

$$(\mathcal{A}, a_1 \cup \cdots \cup a_m).$$

This can be done in PTIME [Don70]. Then accept iff this intersection is nonempty for all type variables in C.

Hardness follows from a log-space reduction from the DR tree automaton intersection problem. Given a sequence of tree automata $M_1, M_2, \ldots, M_k$, where $M_i = (\mathcal{A}_i, S_i)$ and $\mathcal{A}_i$ is the Σ-algebra (A_i, Σ), we construct a constraint set C and a Haskell type context H such that C is satisfiable with respect to H iff $\bigcap_{i=1}^{k} T(M_i)$ is nonempty. Assume the sets of states $A_1, \ldots, A_k$ are disjoint. For all $1 \leq i \leq k$ add to H a set of assumptions for M_i as follows. If $\sigma^{\mathcal{A}_i}(a) = (a_1, \ldots, a_m)$, for $\sigma \in \Sigma_m$, then add to H,

$$a : \forall \alpha_1 \cdots \alpha_m \textbf{ with } a_1 : \alpha_1, \ldots, a_m : \alpha_m \,.\, \sigma(\alpha_1, \ldots, \alpha_m)$$

and for $\sigma \in \Sigma_0$, add $a : \sigma$ for all $a \in \sigma^{\mathcal{A}_i}$. Then with $C = \{S_1 : \alpha, \ldots, S_k : \alpha\}$, we have $\bigcap_{i=1}^{k} T(M_i)$ is nonempty iff C is satisfiable with respect to H. $\square$

Now we consider the complexity of *CS-SAT* when all assumptions for an overloaded identifier, say x, in a Haskell type context are formed by specializing τ_x with unary or nullary type constructors only. An example of this kind of overloading is given in Figure 2. We call such contexts *unary* Haskell type contexts.

Definition 2.2 *A Haskell type context H is a unary Haskell type context if all assumptions for every identifier x of H have the form $x : \tau_x[\alpha_x := \chi_0]$, for some nullary type constructor χ_0, or $x : \forall \beta$ **with** $C \,.\, \tau_x[\alpha_x := \chi(\beta)]$, for some unary type constructor χ.*

Theorem 2.4 *CS-SAT is PSPACE complete for unary Haskell type contexts.*

Proof. Let H be a unary Haskell type context and C a constraint set. Suppose that for each constraint $x : \rho \in C$, $\rho = \tau_x[\alpha_x := \pi]$, for some type π. Let g be a run of $(\mathcal{A}, \{x\})$ on π, where $(\mathcal{A}, \{x\})$ is constructed "on demand" and in polynomial space by Lemma 2.2. As in Theorem 2.3, assign tree automaton $(\mathcal{A}, a)$ to a type variable β in π if $\pi(w) = \beta$ and $g(w) = a$ for some node w. If a variable α in C has occurrences $\alpha_1, \ldots, \alpha_m$, then check whether the languages of their assigned tree automata $(\mathcal{A}, a_1), \ldots, (\mathcal{A}, a_m)$ intersect by checking the emptiness of $(\mathcal{A}, a_1 \cup \cdots \cup a_m)$. This automaton represents a DFA since H is a unary Haskell type context. Thus emptiness can be checked by searching an on-demand construction of it nondeterministically in $log\ 2^{|H|}$ space, or DSPACE($|H|^2$).

Hardness follows from a log-space reduction from the DFA intersection problem [Koz77]. Let $M_1, M_2, \ldots, M_k$ be a sequence of DFAs, where M_i is $(Q_i, \Sigma, \delta_i, q_{0_i}, F_i)$, and the sets of states $Q_1, \ldots, Q_k$ are disjoint. Create a unary Haskell type context H where for all $1 \leq i \leq k$, assumptions are added to H for M_i as follows. For all $q, q' \in Q_i$ and $a \in \Sigma$ such that $\delta_i(q, a) = q'$, add to H,

$$q : \forall \alpha \textbf{ with } q' : \alpha \,.\, a(\alpha)$$

and for all $q \in F_i$ add $q : \epsilon$ where ϵ is a nullary type constructor. Then with $C = \{q_{0_1} : \alpha, \ldots, q_{0_k} : \alpha\}$, $\bigcap_{i=1}^{k} L(M_i)$ is nonempty iff C is satisfiable with respect to H. □

And finally we consider unary Haskell type contexts without recursion.

Theorem 2.5 *CS-SAT is NP hard for nonrecursive, unary Haskell type contexts.*

Proof. The proof is by a PTIME reduction from 3CNF-SAT. Suppose E is a *3CNF* formula with clauses $d_1, \ldots, d_n$ and distinct variables $x_1, \ldots, x_m$. The satisfying truth assignments for each clause d_i are recognizable as unary trees by a DR tree automaton $M_i = (\mathcal{A}_i, S_i)$ that can be constructed in $O(m)$ time. That is, $B_1(B_2(\cdots B_m(\epsilon)\cdots)) \in T(M_i)$ iff the assignment of truth values $B_1, \ldots, B_m$ to $x_1, \ldots, x_m$ respectively satisfies d_i. Assume the sets of states for the n automata are disjoint. Then for all $1 \leq i \leq n$, add to a type context H, the assumption

$$a : \forall\alpha \textbf{ with } b : \alpha \,.\, B(\alpha)$$

if $B^{\mathcal{A}_i}(a) = b$ and $a : \epsilon$ if $a \in \epsilon^{\mathcal{A}_i}$. Then $\{S_1 : \alpha, \ldots, S_n : \alpha\}$ is satisfiable under H iff E is satisfiable, and H is nonrecursive and unary. □

3 Summary

We have observed that it is necessary to simultaneously restrict both recursion in overloadings and the structure of type assumptions if there is to be any hope of solving *CS-SAT* efficiently. Surprisingly, even for the simple kind of overloading allowed in Haskell the problem is hard.

References

[DaM82] Damas, L. and Milner, R., Principal Type Schemes for Functional Programs, *Proc. 9th ACM Symp. on Principles of Prog. Lang.*, pp. 207-212, 1982.

[Don70] Doner, J., Tree acceptors and some of their applications, *J. Computer and System Sciences*, **4**, pp. 406-451, 1970.

[GeS84] Gecseg, F. and Steinby M., Tree Automata, Akademiai Kiado, Budapest Hungary, 1984.

[Has92] Report on the Programming Language Haskell, A Non-strict, Purely Functional Language, Version 1.2, *ACM SIGPLAN Notices*, 27(5), May 1992.

[Koz77] Kozen, D., Lower Bounds for Natural Proof Systems, *Proc. 18th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Long Beach, CA pp. 254-266, 1977.

[Mil78] Milner, R., A Theory of Type Polymorphism in Programming, *J. Computer and System Sciences*, **17**, pp. 348-375, 1978.

[NiPr93] Nipkow, T. and Prehofer, C., Type Checking Type Classes, *Proc. 20th ACM Symp. on Principles of Prog. Lang.*, pp. 409–418, 1993.

[Rey70] Reynolds, J.C., Transformational Systems and the Algebraic Structure of Atomic Formulas, *Machine Intelligence*, **5**, pp. 135-151, 1970.

[Sei94] Seidl, H., Haskell Overloading is DEXPTIME complete, *Information Processing Letters*, **52**(2), pp. 57-60, October 1994.

[Smi91] Smith, G.S., Polymorphic Type Inference for Languages with Overloading and Subtyping, Ph.D. Thesis, Department of Computer Science, Cornell University, Technical Report 91-1230, 1991.

[Smi93] Smith, G.S., Polymorphic Type Inference with Overloading and Subtyping, *Proc. TAPSOFT '93*, *LNCS* **668**, Springer-Verlag, pp. 671-685, 1993.

[Smi94] Smith, G.S., Principal Type Schemes for Functional Programs with Overloading and Subtyping, *Sci. of Comp. Prog.*, **23**,(2-3) pp. 197-226, Dec. 1994.

[VoS91] Volpano, D.M. and Smith, G.S., On the Complexity of *ML* Typability with Overloading, *Proc. 5th Conf. on Functional Programming Languages and Computer Architecture*, *LNCS* **523**, Springer-Verlag, pp. 15-28, 1991.