On the Complexity of ML Typability with Overloading*

Dennis M. Volpano
Geoffrey S. Smith

TR 91-1210
May 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*The authors acknowledge joint support from the NSF and DARPA under grant ASC-88-00465. This paper was presented at the Conference on Functional Programming Languages and Computer Architecture, Cambridge, MA, August 1991.

On the Complexity of ML Typability with Overloading

Dennis M. Volpano and Geoffrey S. Smith[1]

Department of Computer Science

Cornell University

Ithaca, New York 14853 USA

ABSTRACT

We examine the complexity of type checking in an ML-style type system that permits functions to be overloaded with different types. In particular, we consider the extension of the ML type system proposed by Wadler and Blott in the appendix of [WB89], with global overloading only, that is, where the only overloading is that which exists in an initial type assumption set; no local overloading via **over** and **inst** expressions is allowed. It is shown that under a correct notion of well-typed terms, the problem of determining whether a term is well typed with respect to an assumption set in this system is undecidable. We then investigate limiting recursion in assumption sets, the source of the undecidability. Barring mutual recursion is considered, but this proves too weak, for the problem remains undecidable. Then we consider a limited form of recursion called *parametric recursion*. We show that although the problem becomes decidable under parametric recursion, it appears harder than conventional ML typability, which is complete for DEXPTIME [Mai90].

1. Introduction

A rather obvious limitation of the Hindley-Milner type system [Hin69, Mil78, DM82] is that it does not allow an identifier to be *overloaded*, that is, to possess more than one assumption in a type assumption set. For example, we may want equality to possess precisely types $Int \rightarrow Int \rightarrow Bool$ and $Char \rightarrow Char \rightarrow Bool$, but in the Hindley-Milner type discipline, there is no type assumption set from which we can deduce all and only these types for equality. For this reason, languages whose type systems are based on the Hindley-Milner system are forced to avoid overloading altogether, as in Miranda [Tur86], or allow it but fix the set of overloaded operators, as in Standard ML [HMT88].

Wadler and Blott, in the appendix of [WB89], present an extension of the Hindley-Milner type system that incorporates overloading. The system is the basis for the type system of Haskell, a functional programming language aimed at providing a more standardized notation for the functional programming language community [HW90]. But unlike an earlier extension proposed by Kaes [Kae88], their type system has a new form of type, called a *predicated type*, and allows more expressive type assumption sets. The computational consequences of this increased expressiveness are explored in this paper.

The language considered by Wadler and Blott in the design of their type system is core ML [MH88] with two new kinds of expressions, **over** and **inst**, for overloading identifiers locally. The language we consider is just core ML, so all overloading has global scope, and is introduced through an initial type assumption set only. We call Wadler and Blott's type system without the inference rules for **over** and **inst** system WB.

[1] The authors acknowledge joint support from the NSF and DARPA under grant ASC-88-00465. This paper was presented at the Conference on Functional Programming Languages and Computer Architecture, Cambridge MA, August 1991.

2. System WB

Given a set of type variables $\{\alpha, \beta, \gamma, ...\}$ and type constructors $\{\chi$, *Int*, *Char*, *Set*, *List*, *Pair*, ... $\}$, the types of system WB are defined by

Types	τ	$::= \alpha \mid \tau \to \tau' \mid \chi(\tau_1, \ldots, \tau_n)$
Predicated types	ρ	$::= (x :: \tau).\rho \mid \tau$
Type schemes	σ	$::= \forall\alpha.\sigma \mid \rho$

The parentheses of χ are omitted if it has no arguments. The term $(x :: \tau)$ is called a *predicate* and is viewed as a restriction stating that x has type τ. The τ-describable part of a type scheme is referred to as its type part, or body.

2.1. Type assumption sets

Type checking is done in the context of a set of assumptions which bind type information to the free identifiers in an expression. Wadler and Blott also require type assumptions to specify translations of overloaded identifiers, but we omit them here for they are not relevant to our discussion. An assumption set may contain multiple assumptions per identifier, called *instance assumptions*, each of which is designated $::_i$. All types appearing in the instance assumptions for an identifier are specializations of a single type given in an *overload assumption*, designated $::_o$, for the identifier. All assumption sets in WB must be *valid*, a property whose definition depends on a notion of overlapping type schemes.

Definition. (overlap). Two type schemes σ and σ' overlap if there exists a type τ and a valid set of assumptions A such that $\sigma \geq_A \tau$ and $\sigma' \geq_A \tau$. We write $\sigma \# \sigma'$ if σ and σ' do not overlap.

The definition of overlap is given in terms of the instance relation $\geq_A$, but it need not be, for it is equivalent, under a renaming of bound variables, to merely a test for whether the type parts of two schemes are unifiable.

Theorem 2.1. *Two type schemes, each with bound variables not occurring free or bound in the other, overlap if and only if their type parts are unifiable.*

Proof. Let $\sigma = \forall\alpha_1 \cdots \forall\alpha_m.\rho.\tau$ and $\sigma' = \forall\beta_1 \cdots \forall\beta_n.\rho'.\tau'$ such that α_i is not in σ' and β_i is not in σ.

(only if). Suppose σ and σ' overlap. Then there is a valid assumption set A and type γ such that $\sigma \geq_A \gamma$ and $\sigma' \geq_A \gamma$. By the definition of $\geq_A$, then, there are substitutions $S = [\alpha_1 := \tau_1, \ldots, \alpha_m := \tau_m]$ and $S' = [\beta_1 := \tau_1', \ldots, \beta_n := \tau_n']$ such that $\tau S = \gamma$ and $\tau' S' = \gamma$ (we write the application of a substitution to a term in postfix notation). Thus, $\tau S S' = \tau' S S'$, or τ and τ' are unifiable.

(if). Suppose there is a substitution S such that $\tau S = \tau' S = \gamma$. Let A be an assumption set formed by adding to it $x ::_o \forall\alpha.\alpha$ and $x ::_i \Pi$ for each $x :: \Pi$ in $(\rho \cup \rho')S$. $\Pi_i \# \Pi_j$, whenever $\Pi_i \neq \Pi_j$, so A is valid. Since $\tau S = \gamma$ and $A \vdash \rho S$, $\sigma \geq_A \gamma$. Likewise, $\tau' S = \gamma$ and $A \vdash \rho' S$, so $\sigma' \geq_A \gamma$. Thus, σ and σ' overlap.

Therefore we adopt a much simpler test for whether two type schemes overlap, one that only requires renaming their bound variables so that no bound variable of one occurs in the other and then checking to see if their type parts are unifiable.

Definition. (valid assumption set). The empty set is valid. Let A be a valid assumption set, x an identifier that does not appear in A, and σ a type scheme. Then $A, x :: \sigma$ is valid. If $\tau_1, .., \tau_m$ are types and $\sigma_1, \ldots, \sigma_n$ are type schemes such that

$$\sigma \geq_A \sigma_i \text{, for } 1 \leq i \leq n \text{, and}$$
$$\sigma \geq_A \tau_i \text{, for } 1 \leq i \leq m \text{, and}$$
$$\sigma_i \# \sigma_j \text{ for } i \neq j \text{ and } 1 \leq i, j \leq n$$

then

$$A,\ x ::_o \sigma\ ,$$
$$x ::_i \sigma_1\ , \ldots ,\ x ::_i \sigma_n\ ,$$
$$x :: \tau_1\ , \ldots ,\ x :: \tau_m$$

is a valid assumption set.

The types given to each overloaded identifier in a valid assumption set are pairwise nonoverlapping. This restriction guarantees that any type for an overloaded identifier has at most one derivation in WB, thus ensuring that each resolved occurrence of the identifier in an expression has a unique translation.

A predicated type enables one to assert, via an instance assumption, that an identifier has a certain type provided that the identifier itself has some other type. We characterize those assumption sets in which this capability is exploited as *recursive.*

Definition. (recursive assumption set). Let R be a binary relation on the identifiers of an assumption set A such that $g\ R\ h$ if and only if there is an instance assumption about g in A with a predicate involving h (g and h may be the same identifier). Then A is recursive if and only if R^+ is not irreflexive.

For example, the assumptions in Figure 1 form a valid, recursive assumption set. Under the assumptions in this set, it would be type correct to apply predicate eq to values of types such as $List(List(Int))$, $List(Pair(Int,\ Char))$ and so on.

2.2. Inference rules

The inference rules that we regard as part of system WB, for the purpose of this paper, are all but the rules for **over** and **inst** given in the appendix of [WB89]. These include basically the inference rules of Damas and Milner [DM82], and two new rules **PRED** and **REL**, which are given below without translation information.

$$eq ::_o \forall\alpha.\alpha \to \alpha \to Bool$$
$$eq ::_i Int \to Int \to Bool$$
$$eq ::_i Char \to Char \to Bool$$
$$eq ::_i \forall\alpha.(eq :: \alpha \to \alpha \to Bool).List(\alpha) \to List(\alpha) \to Bool$$
$$eq ::_i \forall\alpha.\forall\beta.(eq :: \alpha \to \alpha \to Bool).(eq :: \beta \to \beta \to Bool).$$
$$Pair(\alpha,\ \beta) \to Pair(\alpha,\ \beta) \to Bool$$

Figure 1. A recursive assumption set.

$$\frac{A,\ (x :: \tau) \vdash e :: \rho}{A \vdash e :: (x :: \tau).\rho} \qquad x ::_o \sigma \in A \tag{PRED}$$

$$\frac{A \vdash e :: (x :: \tau).\rho, \quad A \vdash x :: \tau}{A \vdash e : \rho} \qquad x ::_o \sigma \in A \tag{REL}$$

The **PRED** rule allows an assumption about an identifier used to deduce a type for an expression to be shifted from the assumption set into the type of the expression. Eliminating, or releasing, a predicate $(x :: \tau)$ from a type relative to an assumption set A using rule **REL** requires showing $A \vdash x :: \tau$, that is, that the predicate can be *satisfied* with respect to A.

2.3. Well-typed expressions

The ability to move assumptions about overloaded identifiers from an assumption set into the type of an expression via **PRED** calls into question the notion of typability in WB. That is, when is an expression e well typed with respect to a valid assumption set A? The obvious condition is that e is well typed with respect to A if and only if

$$A \vdash e : \sigma, \quad \text{for some type scheme } \sigma. \tag{1}$$

However, this appears incorrect based on the objectives of system WB. In particular, the condition is too weak to force certain terms to be the source of type errors.

For example, let A be a valid, initial assumption set defined by

$$A = \left\{ \begin{array}{l} mult ::_o \forall\alpha.\, \alpha \to \alpha \to \alpha\,, \\ mult ::_i Int \to Int \to Int\,, \\ mult ::_i Float \to Float \to Float\,, \\ c :: Char \end{array} \right\}$$

In this set, *mult* (multiplication) is defined for, or has instances at, types *Int* and *Float* only. Suppose *square* is a function defined as $\lambda x.\ mult\ x\ x$. Wadler and Blott suggest that *square* applied to c should cause a type error under A since *mult* has no instance at type *Char* in A (see pg. 64 of [WB89]). Yet in their type system, according to condition (1), the application is well typed under A because a type scheme can be derived for it from A as follows.

by rules **TAUT** and **COMB**

$A,\ mult :: Char \to Char \to Char,\ \ x :: Char \vdash (mult\ x\ x) :: Char$

by rule **ABS**

$A,\ mult :: Char \to Char \to Char \vdash (\lambda x.\ mult\ x\ x) :: Char \to Char$

by rule **TAUT**

$A,\ mult :: Char \to Char \to Char \vdash c :: Char$

by rule **COMB**

$A,\ mult :: Char \to Char \to Char \vdash ((\lambda x.\ mult\ x\ x)\ \ c) :: Char$

by rule **PRED**

$A \vdash ((\lambda x.\ mult\ x\ x)\ \ c) :: (mult :: Char \to Char \to Char).Char$

So under condition (1), the application is well typed with respect to A, for we are able to give it a type scheme. That is, it does not cause a type error. In order for it to be the source of a type error, a stronger condition is needed to judge whether terms are well typed. A suitable condition is that e is well typed if and only if it can be given a τ type.

Definition. (well-typed expression). An expression e is well typed with respect to A if and only if $A \vdash e : \tau$, for some τ.[2]

Under this condition, the preceding derivation is not enough to show that function *square* applied to c is well typed. The condition forces us to try to eliminate predicate $(mult :: Char \to Char \to Char)$ relative to A, which is impossible since *mult* does not have an instance at type *Char* in A. So the application is regarded as untypable in the context A, giving us the desired type error.

3. WB typability is undecidable

The power of recursion in assumption sets renders the typability problem in WB (determining whether a term is well typed under a given valid assumption set) undecidable.

Theorem 3.1. *Given a valid assumption set A and an expression e, it is undecidable whether e is well typed under A.*

Proof. We reduce PCP (Post's Correspondence Problem [HU79]) to WB typability. The reduction is presented through an example. Recall that an instance of PCP consists of two lists $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ of strings over some alphabet and has a solution if there is a sequence of integers $i_1, i_2, \ldots, i_m$, for $m \geq 1$, such that $x_{i_1} x_{i_2} \cdots x_{i_m} = y_{i_1} y_{i_2} \cdots y_{i_m}$. Suppose that the PCP instance we are given is

$$\begin{array}{ll} x_1 = 10 & y_1 = 101 \\ x_2 = 011 & y_2 = 11 \\ x_3 = 101 & y_3 = 011 \end{array}$$

Assume that there are type constants 0, 1, ε, and t_i, for $1 \leq i \leq 3$, and that $\to$ is right associative. Let A be an assumption set containing all and only the assumptions

$$\begin{array}{l} pcp ::_o \forall\alpha.\alpha \\ pcp ::_i (1 \to 0 \to \varepsilon) \to (1 \to 0 \to 1 \to \varepsilon) \to t_1 \\ pcp ::_i (0 \to 1 \to 1 \to \varepsilon) \to (1 \to 1 \to \varepsilon) \to t_2 \\ pcp ::_i (1 \to 0 \to 1 \to \varepsilon) \to (0 \to 1 \to 1 \to \varepsilon) \to t_3 \\ pcp ::_i \forall\alpha.\forall\beta.\forall\gamma.(pcp :: \alpha \to \beta \to \gamma). \\ \qquad (1 \to 0 \to \alpha) \to (1 \to 0 \to 1 \to \beta) \to (t_1 \to \gamma) \\ pcp ::_i \forall\alpha.\forall\beta.\forall\gamma.(pcp :: \alpha \to \beta \to \gamma). \\ \qquad (0 \to 1 \to 1 \to \alpha) \to (1 \to 1 \to \beta) \to (t_2 \to \gamma) \\ pcp ::_i \forall\alpha.\forall\beta.\forall\gamma.(pcp :: \alpha \to \beta \to \gamma). \\ \qquad (1 \to 0 \to 1 \to \alpha) \to (0 \to 1 \to 1 \to \beta) \to (t_3 \to \gamma) \end{array}$$

By the t_i components of the types of *pcp*, the assumptions do not overlap. Therefore, A is valid. Then the function $\lambda x.\, pcp\ x\ x$ is well typed with respect to A if and only if $A \vdash pcp :: \tau \to \tau \to \gamma$ for some τ and γ. But the assumptions for *pcp* allow it to have only types of the form $\tau \to \tau' \to \gamma$, where τ is obtained by concatenating various x_i's and τ' by concatenating the corresponding y_i's (recall that $\to$ is right associative). Hence $\lambda x.\, pcp\ x\ x$

[2] Though the definition is adequate for the purpose of this paper, it is unsatisfactory for typing let [Smi89]. Actually rule **PRED** should be reformulated, allowing the introduction of only those predicates that are satisfiable.

is well typed with respect to A if and only if the PCP instance has a solution. **Q.E.D.**

We can also show that the instance relation of system WB is undecidable.

Theorem 3.2 . *Given a valid assumption set A and two types* σ *and* σ' *, it is undecidable whether* $\sigma \geq_A \sigma'$.

Proof. Let A be a valid assumption set encoding a PCP instance as in Theorem 3.1. Then

$$\forall\alpha.\,\forall\gamma.\,(pcp :: \alpha \to \alpha \to \gamma).\,int \geq_A int$$

if and only if the PCP instance has a solution. **Q.E.D.**

By Theorem 2.1, the instance relation is no longer needed to determine overlap, but the validity condition still depends on it. All types appearing in the instance assumptions for an identifier must be instances of the type given in the overload assumption. Thus we have the following immediate corollary.

Corollary . *It is undecidable whether an assumption set is valid.*

Proof. Given a valid assumption set A and two types σ and σ', let x be an identifier that does not appear in A. Then

$$A,\ x ::_o \sigma,\ \ x ::_i \sigma'$$

is valid if and only if $\sigma \geq_A \sigma'$. **Q.E.D.**

4. Limiting recursion

In light of Theorem 3.1, we wish to identify a restriction on type assumption sets with which WB typability becomes decidable. WB typability is decidable for nonrecursive assumption sets, but banning recursion altogether seems unacceptable since recursive assumption sets, like the set in Figure 1, arise naturally in practice. So we prefer to restrict it instead. First we consider a restriction that prohibits *mutual recursion*.

Definition. (mutually-recursive assumption set). An assumption set is mutually recursive if and only if it contains a sequence of distinct instance assumptions of the form

$$\begin{aligned} &h_0 ::_i \forall\alpha_1 \cdots \forall\alpha_{m_0}.\,(h_1 :: \tau_0).\,\rho_0 \\ &h_1 ::_i \forall\alpha_1 \cdots \forall\alpha_{m_1}.\,(h_2 :: \tau_1).\,\rho_1 \\ &\qquad\vdots \\ &h_{n-1} ::_i \forall\alpha_1 \cdots \forall\alpha_{m_{n-1}}.\,(h_0 :: \tau_{n-1}).\,\rho_{n-1} \end{aligned}$$

for $n > 1$ (the h_i's need not be distinct identifiers).

At first glance, it appears that limiting assumption sets to sets that are not mutually recursive is overly restrictive. The assumptions in Figure 1, for example, would be illegal. But, surprisingly, enough expressive power has been retained to permit an alternative formulation of them without mutual recursion. In fact, so much has been retained that assumption sets are still too expressive.

Theorem 4.1 . *Given a valid assumption set A that is not mutually recursive and an expression e, it is undecidable whether e is well typed under A.*

Proof. Again we reduce PCP to WB typability. As in the proof of Theorem 3.1, the reduction is presented through an example. Suppose that the PCP instance we are given is the same one given in the proof of Theorem 3.1. Assume that there are type constants 0 , 1 , ε , and t_i , for $1 \leq i \leq 3$, and that $\to$ is right associative. Initially, let A be an assumption set containing only the assumptions $xy_j ::_o \forall\alpha.\alpha$, for $1 \leq j \leq 3$. Then add to A the assumptions

$$xy_1 ::_i (1 \to 0 \to \varepsilon) \to (1 \to 0 \to 1 \to \varepsilon) \to t_1$$
$$xy_2 ::_i (0 \to 1 \to 1 \to \varepsilon) \to (1 \to 1 \to \varepsilon) \to t_2$$
$$xy_3 ::_i (1 \to 0 \to 1 \to \varepsilon) \to (0 \to 1 \to 1 \to \varepsilon) \to t_3$$

For each j, $1 \le j \le 3$, add to A,

$$xy_j ::_i \forall\alpha.\ \forall\beta.\ \alpha \to \beta \to t_k$$

for all k such that $1 \le k \le 3$ and $j \ne k$. Next, add to A the assumptions

$$\begin{aligned}
&\mathit{append} ::_o \forall\alpha.\ \alpha \\
&\mathit{append} ::_i \forall\alpha.\ \varepsilon \to \alpha \to \alpha \\
&\mathit{append} ::_i \forall\alpha.\ \forall\beta.\ \forall\gamma.\ \forall\delta.\ (\mathit{append} :: \alpha \to \beta \to \gamma). \\
&\qquad (\delta \to \alpha) \to \beta \to (\delta \to \gamma)
\end{aligned}$$

The effect of $\mathit{append} : \alpha \to \gamma \to \pi$ is to bind to π, γ appended to the right of α.

The idea is to construct an assumption which effectively selects one of the x_i's and corresponding y_i's for concatenation. To this end, add to A,

$$\begin{aligned}
&\mathit{pcp} ::_o \forall\alpha.\ \alpha \\
&\mathit{pcp} ::_i \forall\gamma.\ \forall\rho.\ \forall\tau. \\
&\qquad\qquad (xy_1 :: \gamma \to \rho \to \tau). \\
&\qquad\qquad (xy_2 :: \gamma \to \rho \to \tau). \\
&\qquad\qquad (xy_3 :: \gamma \to \rho \to \tau). \\
&\qquad \gamma \to \rho \to t_1 \\
&\mathit{pcp} ::_i \forall\alpha.\ \forall\beta.\ \forall\gamma.\ \forall\rho.\ \forall\pi_1.\ \forall\pi_2.\ \forall\tau.\ \forall\sigma. \\
&\qquad\qquad (\mathit{pcp} :: \alpha \to \beta \to \sigma). \\
&\qquad\qquad (\mathit{append} :: \alpha \to \gamma \to \pi_1). \\
&\qquad\qquad (\mathit{append} :: \beta \to \rho \to \pi_2). \\
&\qquad\qquad (xy_1 :: \gamma \to \rho \to \tau). \\
&\qquad\qquad (xy_2 :: \gamma \to \rho \to \tau). \\
&\qquad\qquad (xy_3 :: \gamma \to \rho \to \tau). \\
&\qquad \pi_1 \to \pi_2 \to t_2
\end{aligned}$$

A is valid, but not mutually recursive, and $\lambda x.\ \mathit{pcp}\ x\ x$ is well typed with respect to A if and only if the PCP instance has a solution. **Q.E.D.**

4.1. Parametric recursion

Fortunately, there is a form of recursion under which WB typability is decidable and that allows the kind of mutual recursion exhibited in Figure 1. We call it *parametric recursion.* Before defining it, we need the definition of *parametric overloading*.

Definition. (parametric overloading). An identifier h is parametrically overloaded in an assumption set A if and only if it has an overload assumption in A of the form $h ::_o \forall\alpha.\tau$, for some τ, and for every instance assumption $h ::_i \forall\beta_1 \cdots \forall\beta_n.\rho.\tau'$ in A, where $n \ge 0$,

(a) $\rho = (h :: \tau[\alpha := \beta_1]), \ldots, (h :: \tau[\alpha := \beta_n])$, and

(b) $\tau' = \tau[\alpha := \chi(\beta_1, \ldots, \beta_n)]$, for some type constructor χ.

A desirable property of parametric overloading is its type-constructor property. For each identifier h overloaded this way in an assumption set A with $h ::_o \forall\alpha.\tau$ as its overload assumption, there is a finite set of type constructors that describes precisely the types τ' for which $A \vdash h :: \tau[\alpha := \tau']$. For example, *eq* of Figure 1 is parametrically overloaded as is *mult* in assumption set A defined in Section 2.3.

Definition. (parametric recursion). An assumption set is parametrically recursive if and only if its only source of recursion is that which comes from overloading identifiers parametrically.

The assumptions in Figure 1, for example, form a parametrically-recursive assumption set. Further, note that any nonrecursive assumption set is trivially parametrically recursive, for the definition merely states that if there is recursion then it must be the kind that comes from overloading identifiers parametrically.

WB typability is also decidable under a more flexible form of parametric overloading where equality in (a) is replaced by containment:

(a) $(h :: \tau[\alpha := \beta_1]), \ldots, (h :: \tau[\alpha := \beta_n]) \subseteq \rho$.

This form corresponds more closely to the parametric overloading notion of Kaes [Kae88]. It is needed to express certain assumptions, however it does not preserve the type-constructor property. An assumption for matrix multiplication, for example, has predicates involving the product and sum of matrix elements where the elements themselves may be matrices:

$$\begin{aligned} &mult ::_i \forall\alpha.\,(add :: \alpha \to \alpha \to \alpha).\,(mult :: \alpha \to \alpha \to \alpha).\\ &\qquad Matrix(\alpha) \to Matrix(\alpha) \to Matrix(\alpha) \end{aligned}$$

The type-constructor set property can be exploited so that if every identifier with an overload assumption is parametrically overloaded, then WB typability is, from a complexity standpoint, as hard as but no harder than conventional ML typability, which is complete for DEXPTIME [Mai90]. However, for parametrically-recursive assumption sets, where some identifiers may not be parametrically overloaded, the problem appears harder than conventional ML typability. Though decidable, it is NEXPTIME hard with respect to polynomial-time reduction which implies that it requires exponential time, nondeterministically.

Theorem 4.2 . *Given a valid, parametrically-recursive assumption set A and an expression e, deciding whether e is well typed with respect to A is NEXPTIME hard.*

Proof. To show that every problem in NEXPTIME is reducible to WB typability, for each nondeterministic Turing machine (NTM) M that is time bounded by $2^{p(n)}$, for some polynomial p of the input length n, we give a polynomial-time algorithm that takes as input a string x and produces a set of type assumptions A_x and an expression e such that e is well typed under A_x if and only if M accepts x.

Let M be a $2^{p(n)}$ time-bounded, one-tape NTM. For each input $x = a_1 a_2 \cdots a_n$, the set A_x is constructed as follows.

Based on an encoding of instantaneous descriptions (ID's), assumptions are generated for an identifier, *move*, that describe M's next-move function δ. Suppose the states of M $(q_0, q_1, \ldots)$, its tape symbols $(X_1, X_2, \ldots)$, and the special symbol ε are type constants. Then an ID $X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n$ is encoded as

$$(X_{i-1} \to \cdots \to X_2 \to X_1 \to \varepsilon) \to q \to (X_i \to \cdots \to X_n \to \varepsilon).$$

Initially, let A_x contain only $move ::_o \forall\alpha.\alpha$. There are two possibilities, left (L) and right

(R), for the direction of the tape head in a single move. If $\delta(q, X)$ contains (q', Y, L), for some states q and q' and tape symbols X and Y, then generate

$$move ::_i \forall\alpha.\,\forall\beta.\,\forall\gamma.\,((\gamma \to \alpha) \to q \to (X \to \beta)) \to (\alpha \to q' \to (\gamma \to Y \to \beta))\,.$$

In addition, if X is B, the blank symbol, then include

$$move ::_i \forall\alpha.\,\forall\beta.\,\forall\gamma.\,((\gamma \to \alpha) \to q \to \varepsilon) \to (\alpha \to q' \to (\gamma \to Y \to \varepsilon))\,.$$

Similarly, if $\delta(q, X)$ contains (q', Y, R), then generate

$$move ::_i \forall\alpha.\,\forall\beta.\,(\alpha \to q \to (X \to \beta)) \to ((Y \to \alpha) \to q' \to \beta)$$

and if $X = B$, also include

$$move ::_i \forall\alpha.\,\forall\beta.\,(\alpha \to q \to \varepsilon) \to ((Y \to \alpha) \to q' \to \varepsilon)\,.$$

Next, assumptions are produced that together describe valid computations of M. For each j such that $1 \le j \le p(n)$, generate $c_j ::_o \forall\alpha.\,\alpha$. Then generate $p(n) + 1$ assumptions, one for each of the identifers $c_1, c_2, \ldots, c_{p(n)+1}$:

$$c_1 ::_i \forall\alpha.\,\forall\beta.\,\forall\gamma.(move :: \alpha \to \beta).(move :: \beta \to \gamma).\alpha \to \gamma$$

$$c_k ::_i \forall\alpha.\,\forall\beta.\,\forall\gamma.(c_{k-1} :: \alpha \to \beta).(c_{k-1} :: \beta \to \gamma).\alpha \to \gamma \quad \forall k.\,1 < k \le p(n)$$

$$c_{p(n)+1} :: \forall\alpha.\,\forall\beta.(c_{p(n)} :: \alpha \to \beta).(final :: \beta).\alpha$$

For $1 \le k \le p(n)$, if τ and τ' encode ID's then c_k has type $\tau \to \tau'$ if and only if there is some sequence of exactly 2^k moves of M from τ to τ'. Since M may accept before making $2^{p(n)}$ moves, it is also necessary to generate

$$move ::_i \forall\alpha.\,\forall\beta.(\alpha \to q_f \to \beta) \to (\alpha \to q_f \to \beta)$$

for each final state q_f of M. The effect is to allow all accepting ID's to repeat so that every valid computation of M can be regarded as consisting of exactly $2^{p(n)}$ moves. So we see that if $c_{p(n)+1}$ has type τ, and τ encodes an ID, then there is an accepting ID that is reachable from τ in exactly $2^{p(n)}$ moves. That the ID is accepting is ensured by the predicate involving *final* and its assumptions, which include the overload assumption $final ::_o \forall\alpha.\,\alpha$, and an instance assumption of the form

$$final ::_i \forall\alpha.\,\forall\beta\,.\,\alpha \to q_f \to \beta$$

for each final state q_f of M.

Finally, with q_0 as M's start state, add to A_x the assumptions

$$ID_0 :: \varepsilon \to q_0 \to a_1 \to a_2 \to \cdots\ a_n \to \varepsilon$$

$$eq :: \forall\alpha.\alpha \to \alpha \to Bool$$

where ID_0 corresponds to the initial ID of M on input x. Then $eq(c_{p(n)+1}, ID_0)$ is well typed under A_x if and only if M accepts x. Moreover, A_x is valid, nonrecursive, and can be generated in time proportional to $p(n)$. **Q.E.D.**

Notice that the proof of Theorem 4.2 makes no use of recursion, so this complexity result applies to WB typability under valid, nonrecursive assumption sets as well.

5. Conclusion

Programs that use overloaded identifiers may apply to many types of inputs for they inherit the multiple types of these identifiers, a kind of polymorphism we call bounded polymorphism. Predicated types are useful for expressing the principal types of such programs. However when coupled with overloading in assumption sets they lead to a very powerful form of expression that must be limited if typability is to be decidable (we have been tempted to create a small library of useful functions encoded as type assumptions in system WB such as *append* in the proof of Theorem 4.1). One approach is to limit recursion in assumption sets to parametric recursion.

We have independently developed our own extension of the Hindley-Milner system that incorporates overloading. The extension preserves two important properties of the original system, namely decidable typability and principal types. It has a new type called a *constrained type* which corresponds to a predicated type in system WB but the inference rules for introducing and eliminating them are different as is the instance relation on types. The only restriction on assumption sets in our system is that they be parametrically recursive. This restriction appears reasonable and unobtrusive in practice based on our experience with an implementation of a type inference algorithm for the system. Other restrictions are considered in [Smi89].

References

[DM82] Damas, L. and Milner, R., Principal type-schemes for functional programs. *Proc. 9th Annual ACM Symp. on Principles of Prog. Lang.*, pp. 207-212, January 1982.

[HMT88] Harper, R., Milner, R. and Tofte, M., The definition of Standard ML. Version 2, ECS-LFCS-88-62, University of Edinburgh, August 1988.

[Hin69] Hindley, R., The principal type scheme of an object in combinatory logic. *Trans. Amer. Math. Soc.*, 146, pp. 29-60, December 1969.

[HU79] Hopcroft, J. and Ullman, J., Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

[HW90] Hudak, P. and Wadler, P., Report on the Programming Language Haskell. Version 1.0, Yale University, April 1990.

[Kae88] Kaes, S., Parametric overloading in polymorphic programming languages. In Lecture Notes in Comp. Sci., *Proc. of the 2nd European Symp. on Programming*, 300, pp. 131-144, 1988.

[Mai90] Mairson, H., Deciding ML typability is complete for deterministic exponential time. *Proc. 17th Annual ACM Symp. on Principles of Prog. Lang.*, pp. 382-401, January 1990.

[Mil78] Milner, R., A theory of type polymorphism in programming. *J. Comp. System Sci.*, 17, pp. 348-375, 1978.

[MH88] Mitchell, J. and Harper, R., The essence of ML. *Proc. 15th Annual ACM Symp. on Principles of Prog. Lang.*, pp. 28-46, January 1988.

[Smi89] Smith, G.S., Overloading and bounded polymorphism. TR 89-1054, Department of Computer Science, Cornell University, November 1989.

[Tur86] Turner, D.A., An overview of Miranda. *SIGPLAN Notices*, December 1986.

[WB89] Wadler, P. and Blott, S., How to make ad-hoc polymorphism less ad-hoc. *Proc. 16th Annual ACM Symp. on Principles of Prog. Lang.*, pp. 60-76, January 1989.