Equivalences

Fritz Henglein*
Courant Institute of Mathematical Sciences
New York University
715 Broadway, 7th floor
New York, N.Y. 10012, USA
Internet: henglein@nyu.edu or henglein@rutgers.edu

October 30, 1988

1 Equivalence Results

In this section we define and show log-space equivalence of the following three problems:

1. Typability of ML+ programs
2. Typability of ML+ programs with only one recursive definition and no let-bindings or nested recursive definitions
3. Semi-Unification

ML+ programs are expressions derivable from E in

$$E ::= x|(EE)|\lambda x.E|\mathbf{let}x = E\mathbf{in}E|\mathbf{fix}x.E$$

where x ranges over a given set of variables. The typing rules for ML+ (see appendix) are identical to the ML typing rules [DM82] but for a more general rule for **fix**-expressions.

Semi-unification is a problem akin to unification. The preordering $\leq$ of *subsumption* on first-order terms is defined by $M \leq N$ if there exists a substitution σ such that $\sigma(M) = N$. A system $\{M_{11} = M_{12}, \ldots, M_{k1} = M_{k2}, N_{11} \leq N_{12}, \ldots, N_{l1} \leq N_{l2}\}$ of term equations and term subsumption inequalities is ***semi-unifiable*** if there is a substitution σ such that all the equalities and subsumption statements $\sigma(M_{11}) = \sigma(M_{12}), \ldots, \sigma(M_{k1}) = \sigma(M_{k2}), \sigma(N_{11}) \leq \sigma(N_{12}), \ldots, \sigma(N_{l1}) \leq \sigma(N_{l2})$ hold.

Polymorphic unification, an extension of ordinary unification recently used by Kanellakis and Mitchell to prove type checking in ML PSPACE-hard [KM89], defines a subclass of semi-unification problems. For example, if $M_1[x, \ldots, x]$ is a term with k occurrences of x and if M_2, M_3 are other terms, then the two extended terms $\mathbf{let}x = M_2\mathbf{in}M_1[x, \ldots, x]$ and M_3 are unifiable if and only if the system $\{M_1[x_1, \ldots, x_k] = M_3, x = M_2, x \leq x_1, \ldots, x \leq x_k\}$ is semi-unifiable.

Theorem 1 *The following three problems are log-space equivalent.*

1. *Typability of arbitrary ML+ programs;*

*This research has been supported by the ONR under contract number N00014-85-K-0413.

2. *typability of ML+ programs of the form* **fix**$x.E$ *where* E *is* **let**- *and* **fix**-*free;*

3. *semi-unifiability of arbitrary systems of term equations and subsumption inequalities.*

Proof (Sketch)

We sketch a proof of the two reductions (1) $\Rightarrow$ (3) and (3) $\Rightarrow$ (2). The first of these reductions can be found in [Hen88]. We shall briefly reiterate the outline of that reduction. In the first step we label the nodes of the syntax tree of a given ML+ program with distinct type variables. We then collect a set of equations between these type variables and type expressions of the form $\tau_1 \longrightarrow \tau_2$ from the typing rules (ABS) and (APPL). For every λ-bound variable x, labelled with type variable t, and every occurrence of x, labelled with t', we add the equation $t = t'$. Now, for every **let**- and **fix**-bound variable x, labelled with the type variable t, and every occurrence of that x, labelled with t', we collect all the λ-bound variables and their type labels $t_1, \ldots, t_k$ in whose scope x occurs and add the subsumption inequality $f(t, t_1, \ldots, t_k) \leq f(t', t_1, \ldots, t_k)$; here f is any suitable function symbol. The resulting system of equations and inequalities has the property that it is semi-unifiable if and only if the original ML+ program is typable.

For the second reduction, (3) $\Rightarrow$ (2), let $\{M_{11} = M_{12}, \ldots, M_{k1} = M_{k2}, N_{11} \leq N_{12}, \ldots, N_{l1} \leq N_{l2}\}$ be a system of equations and inequalities with variables $x_1, \ldots, x_k$. From [KM89] we know that every term can be encoded by **fix**- and **let**-free λ-expressions and that there is a λ-expression $=$ that encodes equality between terms. Similarly, tuples $[L_1, \ldots, L_h]$ and tuple selection functions $\bar{i}([L_1, \ldots, L_h]) = L_i$ can be represented by standard constructions. Now, the λ-expression

$$\mathbf{fix} f.\lambda x_1 \ldots x_k.K[M_1, \ldots, M_k][\lambda y_1 \ldots y_k.\bar{1}(f y_1 \ldots y_k) = N_1, \ldots, \lambda y_1 \ldots y_k.\bar{k}(f y_1 \ldots y_k) = N_k]$$

is typable if and only if the original system of equations and inequalities is semi-unifiable.

This theorem shows that

1. type checking for ML+ programs with only a single **fix** and no **let** is already PSPACE-hard;

2. nesting of **fix**-expressions does not make type checking harder (this is in contrast to Mycroft's statement in [Myc84]);

3. **fix**-expressions are at least as expensive as **let**-expressions as far as type checking is concerned;

4. semi-unification captures the combinatorial essence of type checking ML+ programs.

ML+ Typing Rules

ML+ is an extended λ-calculus. The type expressions are given by

$\tau := t \mid \tau \longrightarrow \tau$

t := (type variables)

$\sigma := \tau \mid \forall t.\sigma$

Type expressions derived from τ above are called *monotypes* and the larger set of type expressions derived from σ are *polytypes*. A type assignment is a mapping from λ-calculus variables to type expressions. For detailed definitions of λ-expressions, type expressions, and type assignments we refer to [DM82] and [Myc84] or any number of other papers on type theory.

The canonical type inference system for the ML+ [Myc84] given below. Let A range over type assignments, x over λ-calculus variables, t over type variables, e and e' over expressions, τ and τ' over monotypes, and σ and σ' over polytypes.

$$\text{(TAUT)} \quad A\{x : \sigma\} \supset x : \sigma$$

$$\text{(INST)} \quad \frac{A \supset e : \forall t.\sigma}{A \supset e : \sigma[\tau/t]}$$

$$\text{(GEN)} \quad \frac{\begin{array}{l} A \supset e : \sigma \\ \qquad (t \text{ not free in } A) \end{array}}{A \supset e : \forall t.\sigma}$$

$$\text{(APPL)} \quad \frac{\begin{array}{l} A \supset e : \tau' \rightarrow \tau \\ A \supset e' : \tau' \end{array}}{A \supset (ee') : \tau}$$

$$\text{(ABS)} \quad \frac{A\{x : \tau'\} \supset e : \tau}{A \supset \lambda x.e : \tau' \rightarrow \tau}$$

$$\text{(LET)} \quad \frac{\begin{array}{l} A \supset e : \sigma \\ A\{x : \sigma\} \supset e' : \sigma' \end{array}}{A \supset \mathbf{let} x = e \mathbf{in} e' : \sigma'}$$

$$\text{(FIX-P)} \quad \frac{A\{x : \sigma\} \supset e : \sigma}{A \supset \mathbf{fix} x.e : \sigma}$$

References

[DM82] L. Damas and R. Milner. Principal type schemes for functional programs. In *Proc. 9th Annual ACM Symp. on Principles of Programming Languages*, pages 207–212, Jan. 1982.

[Hen88] F. Henglein. Type inference and semi-unification. In *Proc. ACM Conf. on LISP and Functional Programming*, ACM, ACM Press, July 1988.

[KM89] P. Kanellakis and J. Mitchell. Polymorphic unification and ML typing (extended abstract). In *Proc. 16th Annual ACM Symp. on Principles of Programming Languages*, ACM, January 1989.

[Myc84] A. Mycroft. Polymorphic type schemes and recursive definitions. In *Proc. 6th Int. Conf. on Programming, LNCS 167*, 1984.