Even Normal Forms Can be Hard to Type

John C. Reynolds[1]
Carnegie Mellon University
Pittsburgh, Pennslyvania 15213 U.S.A.

Extended Abstract — December 1, 1989

1. Introduction

Although type inference, in the form of the Hindley-Milner algorithm, has been successfully applied to languages with a simple type discipline, and even with a limited form of polymorphism [3], it has proved to be a refractory problem for richer type disciplines; it has been shown to be undecidable for the intersection type discipline [2, 1], and its status for the polymorphic lambda calculus [6, 5, 9] is a long-standing open question.

This circumstance has raised interest in the possibility of "partial type inference", in which programs are required to contain some type information but less than in conventional "explicitly typed" formalisms. One plausible approach is based on the observation that programmers normally write β-redices in their "sugared" form as **let** definitions, and that such definitions are a natural context for providing type information. This suggests that one might require type information only in **let** definitions or, in terms of an unsugared language, in redices.

For example, the expression

$$\big(\lambda H.\, H(\lambda k.\, k \times k)\big)\big(\lambda f.\, f(1) + f(2)\big)$$

would normally be written as

$$\mathbf{let}\, H \equiv \lambda f.\, f(1) + f(2)\, \mathbf{in}\, H(\lambda k.\, k \times k)\,,$$

whose partially typed form might be

$$\mathbf{let}\, H\colon (\mathbf{int} \to \mathbf{int}) \to \mathbf{int} \equiv \lambda f.\, f(1) + f(2)\, \mathbf{in}\, H(\lambda k.\, k \times k)\,,$$

[1]Research supported by NSF Grant CCR-8620191. A portion of the research was also sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, under contract number F33615-87-C-1499, monitored by the Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, Ohio. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any agency of the US Government.

whose desugared from is

$$\Big(\lambda H\colon(\mathbf{int}\to\mathbf{int})\to\mathbf{int}\,.\,H(\lambda k.\,k\times k)\Big)\Big(\lambda f.\,f(1)+f(2)\Big)\,.$$

(This brief description is oversimplified. Actually, one requires type information in a kind of generalized redex, called a *cut* by Statman [11], that is the desugaring of a multiple **let** definition.)

However, this approach implies that no type information is needed when an expression contains no redices. Thus its feasibility requires that full type inference be feasible for normal forms. (Throughout this paper, we use "redex" and "normal form" to mean "β-redex" and "β-normal form".) The importance of type inference for normal forms is also suggested by recent work in [10, 4].

The problem of type inference for normal forms will be examined in this paper for the intersection type discipline and for the polymorphic lambda calculus. In the first case we will show that the problem is PSPACE-hard, in the second case we will show that it is just as hard as the type inference problem for arbitrary expressions.

2. Intersection Types

In the intersection type discipline [2, 1], the set of *types* can be defined by the abstract grammar

$$\theta ::= \rho \mid \omega \mid \theta\to\theta \mid \theta\,\&\,\theta$$

where ρ ranges over some set of primitive types and ω is a particular "nonsense" type.

A *type assignment* is a function from a finite set of variables into types. A *typing* is a formula $\pi \vdash e : \theta$, where π is a type assignment, e is an expression of the untyped lambda calculus, and θ is a type. Intuitively, $\pi \vdash e : \theta$ asserts that the expression e has the type θ when its free variables are assigned types by π.

A typing is valid if and only if it can be proved from the following inference rules:

- Variables

$$\frac{}{\pi \vdash x : \pi(x)} \quad \text{when } x \in \operatorname{dom}\pi$$

- Arrow Introduction

$$\frac{[\,\pi \mid x{:}\theta\,] \vdash e : \theta'}{\pi \vdash (\lambda x.\,e) : \theta\to\theta'}$$

- Arrow Elimination

$$\frac{\pi \vdash e_1 : \theta\to\theta' \qquad \pi \vdash e_2 : \theta}{\pi \vdash e_1\,e_2 : \theta'}$$

- Nonsense

$$\frac{}{\pi \vdash e : \omega}$$

- Intersection Introduction

$$\frac{\pi \vdash e : \theta_1 \quad \pi \vdash e : \theta_2}{\pi \vdash e : \theta_1 \,\&\, \theta_2}$$

- Intersection Elimination

$$\frac{\pi \vdash e : \theta_1 \,\&\, \theta_2}{\pi \vdash e : \theta_1} \qquad \frac{\pi \vdash e : \theta_1 \,\&\, \theta_2}{\pi \vdash e : \theta_2}$$

In the rule for arrow introduction, $[\pi \mid x{:}\theta]$ denotes the type assignment whose domain is $\mathrm{dom}\,\pi \cup \{x\}$, such that $[\pi \mid x{:}\theta]x = \theta$ and $[\pi \mid x{:}\theta]x' = \pi x'$ when $x' \neq x$.

Proposition 1 *For the intersection type discipline, deciding the validity of $\pi \vdash e : \theta$, where e is in normal form, is PSPACE-hard.*

The proof is obtained by reducing the problem of evaluating quantified Boolean expressions, which is known to be PSPACE-complete [7], to the type inference problem. The reduction is obtained by translating a quantified Boolean expression B into a lambda expression B^* as follows:

$$(B_1 \wedge B_2)^* = \mathit{and}\, B_1^*\, B_2^*$$
$$(B_1 \vee B_2)^* = \mathit{or}\, B_1^*\, B_2^*$$
$$(\neg B)^* = \mathit{not}\, B^*$$
$$((\forall x)B)^* = \mathit{forall}(\lambda x.\, B^*)$$
$$((\exists x)B)^* = \mathit{exists}(\lambda x.\, B^*)$$
$$x^* = x\ ,$$

where *and*, *or*, *not*, *forall*, and *exists* are variables not occurring in the original expression. In addition a truth value b is translated into a type b^* by

$$\mathit{true}^* = t \qquad\qquad \mathit{false}^* = f\ ,$$

where t and f are distinct types, each of which is either primitive or of the form $\theta_1 \to \theta_2$. Then B evaluates to b under a truth-value assignment β if and only if $\pi \vdash B^* : b^*$ is valid, where π is the type assignment that maps each variable x occurring free in B into $(\beta x)^*$ and maps the additional variables of B^* as follows:

$$\pi(\mathit{and}) = \Big(f \to (f \to f)\Big) \,\&\, \Big(f \to (t \to f)\Big) \,\&\, \Big(t \to (f \to f)\Big) \,\&\, \Big(t \to (t \to t)\Big)$$

$$\pi(\mathit{or}) = \Big(f \to (f \to f)\Big) \,\&\, \Big(f \to (t \to t)\Big) \,\&\, \Big(t \to (f \to t)\Big) \,\&\, \Big(t \to (t \to t)\Big)$$

$$\pi(not) = (f \to t)\,\&\,(t \to f)$$

$$\begin{aligned}\pi(forall) = &\Big(((f \to f)\,\&\,(t \to f)) \to f\Big)\,\&\,\Big(((f \to f)\,\&\,(t \to t)) \to f\Big)\\ &\&\Big(((f \to t)\,\&\,(t \to f)) \to f\Big)\,\&\,\Big(((f \to t)\,\&\,(t \to t)) \to t\Big)\end{aligned}$$

$$\begin{aligned}\pi(exists) = &\Big(((f \to f)\,\&\,(t \to f)) \to f\Big)\,\&\,\Big(((f \to f)\,\&\,(t \to t)) \to t\Big)\\ &\&\Big(((f \to t)\,\&\,(t \to f)) \to t\Big)\,\&\,\Big(((f \to t)\,\&\,(t \to t)) \to t\Big)\,.\end{aligned}$$

Although Proposition 1 deals with the problem of type inference, a similar result holds for type checking, at least for one view of what constitutes explicit typing. In [8], an explicitly typed version of the lambda calculus with intersection types is obtained by replacing the rule for arrow introduction by

- Explicitly Typed Arrow Introduction

$$\frac{[\,\pi \mid x{:}\theta_i\,] \vdash e : \theta'}{\pi \vdash (\lambda x{:}\theta_1, \ldots, \theta_n.\ e) : \theta_i \to \theta'}$$

which limits the types assigned to a bound variable to members of a list stated explicitly in the binding expression. It is easily seen that Proposition 1 continues to hold for this variant of the lambda calculus; one simply replaces $\lambda x.$ by $\lambda x{:}f,t.$ in the translations of quantified Boolean expressions.

3. The Polymorphic Lambda Calculus

In the polymorphic lambda calculus [6, 5, 9], the set of types is defined by

$$\theta ::= \rho \mid \tau \mid \theta \to \theta \mid \forall\tau.\ \theta$$

where ρ ranges over some set of primitive types and τ ranges over a denumerably infinite set of *type variables*. Type assignments and typings are defined as before.

A typing is valid if and only if it can be proved from the following inference rules:

- Variables

$$\frac{}{\pi \vdash x : \pi(x)} \quad \text{when } x \in \operatorname{dom}\pi$$

- Arrow Introduction

$$\frac{[\,\pi \mid x{:}\theta\,] \vdash e : \theta'}{\pi \vdash (\lambda x.\ e) : \theta \to \theta'}$$

- Arrow Elimination

$$\frac{\pi \vdash e_1 : \theta \to \theta' \quad \pi \vdash e_2 : \theta}{\pi \vdash e_1\, e_2 : \theta'}$$

- Quantifier Introduction

$$\frac{\pi \vdash e : \theta}{\pi \vdash e : \forall \tau.\, \theta} \quad \text{when } \tau \text{ not free in } \pi x \text{ for any } x \in \operatorname{dom} \pi$$

- Quantifier Elimination

$$\frac{\pi \vdash e : \forall \tau.\, \theta}{\pi \vdash e : (\theta / \tau \to \theta')}$$

In the last rule, $(\theta/\tau \to \theta')$ denotes the result of substituting θ' for the free occurrences of τ in θ, with renaming of type variables when necessary. Note that, since the expressions e contain no type information, we are dealing with the implicitly typed polymorphic lambda calculus.

Proposition 2 *For the implicitly typed polymorphic lambda calculus, deciding the validity of $\pi \vdash e : \theta$, where e is in normal form, is just as hard as deciding the validity of $\pi \vdash e : \theta$ for arbitrary e.*

The proof is obtained by reducing the type inference problem for arbitrary expressions to the type inference problem for normal forms. Suppose e is an arbitrary expression, i is an identifier not occurring in e, and e^* is obtained from e by replacing each redex $(\lambda x.\, e_1)e_2$ by $\Big(i(\lambda x.\, e_1)\Big)e_2$. Then e^* is in normal form, and $\pi \vdash e : \theta$ is valid if and only if

$$[\, \pi \mid i{:}\forall \tau.\, (\tau \to \tau)\,] \vdash e^* : \theta$$

is valid.

Acknowledgements The author wishes to thank Mark Lillibridge for pointing out that the evaluation of quantified Boolean expressions is PSPACE-complete.

References

[1] Coppo, M., Dezani-Ciancaglini, M., Honsell, F., and Longo, G. *Extended Type Structures and Filter Lambda Models.* in: **Logic Colloquium '82**, Florence, Italy, August 23–28, 1982, edited by G. Lolli, G. Longo, and A. Marcja. **Studies in Logic and the Foundations of Mathematics**, vol. 112, North-Holland, Amsterdam, 1984, pp. 241–262.

[2] Coppo, M., Dezani-Ciancaglini, M., and Venneri, B. *Functional Characters of Solvable Terms.* **Zeitschrift für Mathematische Logik und Grundlagen der Mathematik**, vol. 27 (1981), pp. 45–58.

[3] Damas, L. and Milner, R. *Principal Type-Schemes for Functional Programs.* in: **Conference Record of the Ninth Annual ACM Symposium on Principles of Programming Languages**, Albuquerque, New Mexico, January 25–27. 1982, pp. 207–212.

[4] Giannini, P. and Ronchi Della Rocca, S. *Characterization of Typings in Polymorphic Type Discipline.* in: **Proceedings Third Annual Symposium on Logic in Computer Science**, Edinburgh, Scotland, July 5–8. 1988, pp. 61–70.

[5] Girard, J.-Y. *Interprétation Fonctionnelle et Élimination des Coupures de l'Arithmétique d'Ordre Supérieur*, Thèse de doctorat d'état. Université Paris VII, June 1972.

[6] Girard, J.-Y. *Une Extension de l'Interprétation de Gödel à l'Analyse, et son Application à l'Élimination des Coupures dans l'Analyse et la Théorie des Types.* in: **Proceedings of the Second Scandinavian Logic Symposium**, University of Oslo, June 18–20, 1970, edited by J. E. Fenstad. **Studies in Logic and the Foundations of Mathematics**, vol. 63, North-Holland, Amsterdam, 1971, pp. 63–92.

[7] Hopcroft, J. E. and Ullman, J. D. **Introduction to Automata Theory, Languages, and Computation**. Addison-Wesley, Reading, Massachusetts, 1979, x+418 pp.

[8] Reynolds, J. C. *Preliminary Design of the Programming Language Forsythe.* Report, no. CMU–CS–88–159, Carnegie Mellon University, Computer Science Department, June 21, 1988.

[9] Reynolds, J. C. *Towards a Theory of Type Structure.* in: **Programming Symposium**, Proceedings, Colloque sur la Programmation, Paris, April 9–11, edited by B. Robinet. **Lecture Notes in Computer Science**, vol. 19, Springer-Verlag, Berlin, 1974, pp. 408–425.

[10] Ronchi Della Rocca, S. *Principal Type Scheme and Unification for Intersection Type Discipline.* **Theoretical Computer Science**, vol. 59 (1988), pp. 181–209.

[11] Statman, R. *Normal Varieties of Combinators.* Report, no. 89–61, Carnegie Mellon University, Department of Mathematics, October 1989.