INFORMATION AND CONTROL 22, 257–267 (1973)

The Undecidability of Unification in Third Order Logic*

GERARD P. HUET

Computing and Information Sciences Department, Case Western Reserve University, Cleveland, Ohio 44106

The problem of the existence of a unifying substitution between two terms is considered in type theory. This problem is shown to be undecidable, even if we restrict the objects of the language to third order. This means that we are not able to recognize whether two terms have a common instance or not. This result has important implications for the mechanization of higher order logic.

INTRODUCTION

The unification problem in a logic $\mathscr{L}$ is the following: given two well formed formulas e_1 and e_2 of $\mathscr{L}$, decide whether there exists a substitution σ for the three variables of e_1 and e_2 such that $\sigma \circ e_1$ and $\sigma \circ e_2$ are identical well formed formulas.

The unification problem is decidable in first order logic. It has been independently investigated by J. R. Guard (1964) under the name of "matching" and J. A. Robinson (1965). This last paper contains the description of an algorithm which, in case of a success, returns a unique substitution called the most general unifier (MGU) for e_1 and e_2. This MGU σ has the property that every unifier τ for e_1 and e_2 is a particular instance of σ, i.e., there exists a substitution ρ such that:

$$\tau = \rho \circ \sigma.$$

The existence of this MGU is of critical importance for the proof procedures currently used in automatic theorem proving. Basically it permits us to restrict the rule of substitution to the most general substitution permitting an application of the cut rule (i.e., modus ponens). In other words, we analyze

* The work reported in this paper was supported by the National Science Foundation under Grant No. GJ-1135.

the possibility of cutting two literals $L1$ and $\neg L2$ by computing the MGU of $L1$ and $L2$ (in the general case we try to unify sets of literals). This method, called resolution, has been described as a refutation procedure to detect the unsatisfiability of a set of formulas and proven complete by Robinson (1965).

Many workers in automatic theorem proving have argued after J. A. Robinson (1969) that we should mechanize higher order logic. Toward this goal, P. B. Andrews (1971) has described a refutation system for Church's type theory. However, the substitution rule is still explicit in this system, and its elimination and replacement by a resolution-like rule would involve the computation of unifiers. This task presents many new difficulties in higher order logic. W. E. Gould (1966) has shown that certain pairs of terms do not possess a MGU, and that there may even exist an infinity of independent unifiers. He still conjectured the unification problem to be solvable. This paper disproves Gould's conjecture and shows that, even in third order logic, it is not possible to recognize when two terms have a common instance.

The system of logic used in the proof is described in Section 1. The reader is assumed to be familiar with the λ-calculus notation. Section 2 states the Post correspondence problem over a two letter alphabet, and Section 3 proves its equivalence with a special case of the unification problem in our logic.

1. The Simple Theory of Types

We describe in this section the language of our logic and give the main definitions. Basically, we shall use a slight modification of the system of Church (1940). In particular, we shall replace constructs of the form $[\cdots[[f(x_1)](x_2)]\cdots](x_n)$ by constructs $f(x_1, x_2, \ldots, x_n)$.

1.1. *Types*

Every well formed expression of the language possesses a *type*, which indicates its position in a functional hierarchy.

We suppose that there exists a finite set T_0 of elementary types. The set T of types is generated by the following recursive definition:

(1) $T_0 \subset T$;

(2) $t_1, t_2, \ldots, t_n \in T \,\&\, t \in T_0 \supset (t_1, t_2, \ldots, t_n \rightarrow t) \in T$, for any $n > 0$.

In the examples we shall usually assume the existence of two elementary types: ι for "individuals" and o for "truth values."

1.2. *Terms*

The set of well formed expressions, or *terms*, consists of atoms, applications, and abstractions. Every term e possesses a type $\tau(e)$ in T, given by the mapping τ defined recursively below.

1.2.1. *Atoms*

There exist a denumerable set of variables of each type, and an arbitrary number of constants of any type. They constitute the set of *atoms*. We suppose that variables and constants are distinct, and that atoms of different types are distinct. This permits us to avoid subscripting the atoms with their type. We shall usually denote variables by lower case letters and constants by capitals.

1.2.2. *Applications*

If e is a term of type

$$\tau(e) = (t_1 , t_2 ,..., t_n \rightarrow t), \qquad n \geqslant 1,$$

and if $e_1 , e_2 ,..., e_m$ are terms of types

$$\tau(e_i) = t_i \qquad 1 \leqslant i \leqslant m \leqslant n,$$

then we define the *application*

$$e' = e(e_1 , e_2 ,..., e_m),$$

as a term of type:

$$\tau(e') = \begin{cases} (t_{m+1} , t_{m+2} ,..., t_n \rightarrow t), & \text{if } n > m, \\ t, & \text{if } n = m. \end{cases}$$

Such a term represents the object resulting from the application of the function represented by e to m arguments represented by the e_i's. This object will itself be a function if $n > m$.

1.2.3. *Abstractions*

If e is a term and $u_1 , u_2 ,..., u_n$ are distinct variables $n \geqslant 1$, with

$$\tau(e) = t,$$

and

$$\tau(u_i) = t_i \qquad 1 \leqslant i \leqslant n,$$

then we define the *abstraction*

$$e' = (\lambda u_1 u_2 \cdots u_n) \cdot e$$

as a term of type

$$\tau(e') = \begin{cases} (t_1, t_2, \ldots, t_n \to t), & \text{if } t \in T_0, \\ ((t_1, t_2, \ldots, t_n, t_1', t_2', \ldots, t_m' \to t'), & \text{if } t = (t_1', t_2', \ldots, t_m' \to t'). \end{cases}$$

Such a term represents the function which, when applied to arguments represented by terms $e_1, e_2, \ldots, e_n$, will return as result the object represented by term e, where every occurrence of variable u_i is replaced by e_i, $1 \leqslant i \leqslant n$.

Every occurrence of variable u_i in e' is said to be *bound* in e'. Any occurrence of a variable which is not bound in some subterm of a term e is said to be *free* in e. A term without any free occurrences of variables is said to be *closed.*

In the following, we shall write $[e/u]E$ for the term obtained from E by replacing every occurrence of variable u in E by an occurrence of term e. We impose of course $\tau(e) = \tau(u)$. We want next to define the operation of substitution of a term for a free variable. First we define the normal form of a term so as to make this definition easy.

1.2.4. *Normal Form*

We suppose that for each type α, the set of variables $V(\alpha)$ of type α is ordered by some alphabetic ordering. Let E by any term and $F(E)$ the set of variables having some free occurrence in E. We define:

$$S(E, \alpha) = V(\alpha) - F(E).$$

Now we define the normal form of E as the result of applying to E the following rules in their order of listing until no further reduction is possible:

(1) Rename all bound variables of type α in E by members of $S(E, \alpha)$ so that variables bound by different λ's are given different names, and the order from left to right of bound variables of type α in the resulting formula is a proper initial segment of $S(E, \alpha)$.

(2) Replace any subterm of the form

$$((\lambda u_1 u_2 \cdots u_n) \cdot e)(e_1', e_2', \ldots, e_m'), \quad n, m \geqslant 1 \quad \text{by:}$$

$$\begin{aligned} &- [e_1'/u_1][e_2'/u_2] \cdots [e_n'/u_n]\, e(e_{n+1}', \ldots, e_m'), && \text{if } m > n, \\ &- [e_1'/u_1][e_2'/u_2] \cdots [e_n'/u_n] e, && \text{if } m = n, \\ &- [e_1'/u_1][e_2'/u_2] \cdots [e_m'/u_m](\lambda u_{m+1} \cdots u_n) \cdot e, && \text{if } m < n. \end{aligned}$$

(3) Replace any subterm of the form

$$(e(e_1, e_2, \ldots, e_n))(e_1', e_2', \ldots, e_m'), \qquad n, m \geqslant 1,$$

by

$$e(e_1, e_2, \ldots, e_n, e_1', \ldots, e_m').$$

(4) Replace any subterm of the form

$$(\lambda u_1 u_2 \cdots u_n) \cdot ((\lambda u_1' u_2' \cdots u_m') \cdot e), \qquad n, m \geqslant 1,$$

by

$$(\lambda u_1 u_2 \cdots u_n\, u_1' \cdots u_m') \cdot e.$$

It can be proved (Andrews, 1971; Pietrzykowski, 1971) that, by this reduction process, every term is ultimately reduced into a unique normal form:

$$(\lambda u_1 u_2 \cdots u_n) \cdot F(e_1, e_2, \ldots, e_m),$$

where $n, m \geqslant 0$, $u_1, u_2, \ldots, u_n$ are distinct variables, F is an atom and $e_1, e_2, \ldots, e_m$ are terms. (We delete the corresponding parenthesis if $n = 0$ or $m = 0$.) F is called the *head* of the term.

From now on two terms will be considered identical if they have the same normal form.

1.3. *Substitution*

A *substitution* σ is a set of ordered pairs

$$\{\langle u_i, e_i\rangle \mid 1 \leqslant i \leqslant n\},$$

where the u_i's are distinct variables, and $\tau(e_i) = \tau(u_i)$ $1 \leqslant i \leqslant n$. The *application* of σ to any term E, written $\sigma \circ E$, is defined as the normal form of:

$$((\lambda u_1 u_2 \cdots u_n) \cdot E)(e_1, e_2, \ldots, e_n).$$

It should be noted that possible conflicts of variables are taken care of automatically by our definition of normal form. Therefore $\sigma \circ E$ does not depend on the order in which we take the pairs in σ.

1.4. *Unification*

Two terms e_1 and e_2 of the same type are said to be *unifiable* if there exists a substitution σ, called a *unifier* for e_1 and e_2, such that $\sigma \circ e_1 = \sigma \circ e_2$.

We are assuming here that if some variable appears free in both e_1 and e_2, it represents the same object. We need this formulation if we want to factor two literals in the same clause. To resolve literals of different clauses however, we should have first to rename duplicated free variables. (Remember that

free variables are in fact bound by universal quantifiers at the level of the clause, see Robinson (1965).)

1.5. *Notation*

We shall use in this chapter the following variables:

s, t	of type	$\iota \in T_0$,
u, v, w_n	of type	$(\iota \to \iota) \quad n \geqslant 1$,
g	of type	$((\iota \to \iota) \to \iota)$,
h	of type	$(\iota, \iota \to \iota)$,
f_n	of type	$((\iota \underset{1}{\to} \iota), (\iota \underset{2}{\to} \iota), \ldots, (\iota \underset{n}{\to} \iota) \to \iota) \quad n \geqslant 1.$

Let us define the *order* of an atom as the depth of nesting of parentheses in its type $+1$. Then s is of order one, u is of order two and g is of order three. If we define the order of a language as the maximum order of its variables, we need here a language of order three or more.

2. Post Correspondence Problem

2.1.

Let $\Sigma = \{U, V\}$. A Post problem over Σ is defined by a set of pairs $\{(x_i, y_i) \mid x_i, y_i \in \Sigma^+, 1 \leqslant i \leqslant n\}$. A solution to this Post problem is a sequence $i_1, i_2, \ldots, i_p$, $(p \geqslant 1)$ of integers between 1 and n such that

$$x_{i_1} x_{i_2} \cdots x_{i_p} = y_{i_1} y_{i_2} \cdots y_{i_p}.$$

The general problem of deciding whether a Post problem admits a solution is recursively unsolvable, as shown in Post (1946).

2.2. *Definition*

The Post problem pertains to words over the alphabet Σ, whereas our language deals with functional objects. Let us associate with Σ the set of function variables $\{u, v\}$. With every word x in Σ^* we can associate a term $\tilde{x}$ of type $(\iota \to \iota)$ according to the following definition:

$$\tilde{\epsilon} = (\lambda t) \cdot t,$$
$$\widetilde{Uz} = (\lambda t) \cdot u(\tilde{z}(t)),$$
$$\widetilde{Vz} = (\lambda t) \cdot v(\tilde{z}(t)).$$

This correspondence is obviously a bijection. Now, to the concatenation of words over Σ we make correspond the composition of the corresponding unary functions in our language.

3. Theorem

Theorem. *The problem of the existence of a unifier for two terms in type theory is recursively undecidable.*

Proof. With any Post problem $\{(x_i , y_i) | 1 \leqslant i \leqslant n\}$ over Σ, we associate the unification problem for the two terms:

$$e_1 = (\lambda uvh) \cdot h(f_n(\tilde{x}_1 , \tilde{x}_2 ,..., \tilde{x}_n), f_n(u, u,..., u)),$$

and

$$e_2 = (\lambda uvh) \cdot h(f_n(\tilde{y}_1 , \tilde{y}_2 ,..., \tilde{y}_n), u(g(u))).$$

We prove that these two problems are equivalent by showing that a solution to one implies a solution to the other.

3.1.

Let us suppose that there exists a solution $i_1 , i_2 ,..., i_p , p \geqslant 1$, to the Post problem. The following substitution is a unifier for e_1 and e_2 :

$$\sigma = \{\langle f_n , (\lambda w_1 w_2 \cdots w_n) \cdot w_{i_1}(w_{i_2}(\cdots(w_{i_p}(s))\cdots))\rangle,$$
$$\langle g, (\lambda u) \cdot \underbrace{u(u(\cdots(u}_{p-1}(s))\cdots))\rangle\}.$$

Proof. Noting $z = x_{i_1} x_{i_2} \cdots x_{i_p} = y_{i_1} y_{i_2} \cdots y_{i_p}$, we get:

$$\sigma \circ e_1 = \sigma \circ e_2 = (\lambda uvh) \cdot h(\tilde{z}(s), u^p(s)).$$

3.2.

Let us suppose now that we have a solution σ to our unification problem. Without loss of generality, we can assume that σ is given as:

$$\sigma = \{\langle f_n , e_f\rangle, \langle g, e_g\rangle\},$$

where u, v, and h do not appear in e_f or e_g . For example, if u appeared free in e_f , by our definition of substitution the bound u, v and h in e_1 and e_2 would get renamed into say u', v' and h', and the following discussion would go through by replacing consistently u by u', v by v' and h by h'. If it appeared

bound, it would itself get renamed for the same reason. These considerations permit us to assert that:

$$\sigma \circ e_1 = \sigma \circ e_2 = (\lambda uvh) \cdot h(E_1, E_2),$$

with

$$E_1 = e_f(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n) = e_f(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_n).$$
$$E_2 = e_f(u, u, \ldots, u) = u(e_g(u)).$$

We shall ignore E_2 for the moment (its only purpose is to avoid the trivial solution $p = 0$ to the Post problem) and consider the structure of E_1. Let us suppose that e_f is given (in normal form) as:

$$e_f = (\lambda w_1 w_2 \cdots w_k) \cdot e,$$

and let A be its head (i.e., $e = A(\cdots)$).

Because $\tau(e_f) = \tau(f_n)$, we must have:

$$\cdot\, k \leqslant n,$$

$$\cdot\, \tau(A) = (\alpha_1, \alpha_2, \ldots, \alpha_m, \underbrace{(\iota \to \iota), \ldots, (\iota \to \iota)}_{n-k} \to \iota),$$

for some $\alpha_1, \alpha_2, \ldots, \alpha_m \in T$, $m \geqslant 0$.

We have two cases:

3.2.1. $0 \leqslant k < n$.

Because of its type, A cannot be any w_j $1 \leqslant j \leqslant n$. Therefore, E_1 (as $e_f(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$) will get reduced to a term in normal form such as:

$$A(B_1, B_2, \ldots, B_m, \tilde{x}_{k+1}, \ldots, \tilde{x}_n).$$

As it must also be the normal form of $e_f(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_n)$, we must have $\tilde{x}_j = \tilde{y}_j$ for $k < j \leqslant n$, which forces $x_j = y_j$ $k < j \leqslant n$. In this case the Post problem will admit as solution any sequence of integers between $k + 1$ and n.

3.2.2. $k = n$.

Now A may be a $w_j : e = w_{i_1}(B(\cdots))$; B in turn may be a w_j, etc. Let us define p as the maximum integer such that

$$e = w_{i_1}(w_{i_2}(\cdots(w_{i_p}(e'))\cdots)).$$

That is, $p \geqslant 0$, $w_{i_1}, \ldots, w_{i_p} \in \{w_j \mid 1 \leqslant j \leqslant n\}$, and the head of e' is not a w_j. Now, after reduction to normal form, we shall get:

$$E_1 = \tilde{x}_{i_1}(\tilde{x}_{i_2}(\cdots(\tilde{x}_{i_p}(e''))\cdots)) = \tilde{y}_{i_1}(\tilde{y}_{i_2}(\cdots(\tilde{y}_{i_p}(e'''))\cdots)).$$

Claim. The head of e'' is neither u nor v.

Proof. By hypothesis, u and v do not appear in e'. The only way to introduce them in E_1 is via an argument $\tilde{x}_j$, and by definition of p this is not the case for the head of e''. Thus this atom is the first of E_1 which is not a u or a v. By symmetry it is therefore also the head of e'''. Now consider the initial part of E_1 up to this atom, remove all parentheses and change u in U and v in V. Clearly:

$$x_{i_1}x_{i_2}\cdots x_{i_p} = y_{i_1}y_{i_2}\cdots y_{i_p}.$$

Claim. $p > 0$.

Proof. Assume $p = 0$. For the same reason as in the previous proof, $E_2 = e_f(u, u,..., u)$ is not headed by a u. As it is also $u(e_g(u))$ we get a contradiction. This concludes the proof that $i_1, i_2,..., i_p$ is a solution to the Post problem.

3.1 and 3.2 prove the equivalence of the two problems, which proves the recursive unsolvability of the unification problem.

3.3. *Remarks*

3.3.1. We need in this proof to have the same variable f_n in both e_1 and e_2. If we considered free variables independent in e_1 and e_2, the same proof would go through with the following terms:

$$e_1' = (\lambda uvh') \cdot h'(f_n, f_n(\tilde{x}_1,..., \tilde{x}_n), f_n(u,..., u)),$$

and

$$e_2' = (\lambda uvh') \cdot h'(f_n', f_n'(\tilde{y}_1,..., \tilde{y}_n), u(g(u))),$$

f_n' and h' being of appropriate types.

3.3.2. We could replace the bound variable h in e_1 and e_2 by a constant H, but we prefer not to make any assumptions on the existence of certain constants in the logic. Our proof is valid in any usual formulation of type theory. For example, in Church's system, e_1 would be written as:

$$(\lambda u)[(\lambda v)[(\lambda h)[h([\cdots[[f_n(\tilde{x}_1)](\tilde{x}_2)]\cdots](\tilde{x}_n))]([\cdots[[f_n(u)](u)]\cdots](u))]]$$

3.3.3. Finally, we note that we used in the proof variables of order at most 3 (g and f_n). Therefore the proof still holds if we restrict our language to order three. A similar result has been found independently by C. Lucchesi (1972).

Unification is known to be decidable in languages of first order, as shown in Robinson (1965). The decidability or undecidability of unification for second order logic is still an open question.

Conclusion

We have shown in Section 3 that it is not decidable whether two terms of order three or more have a common instance. This result has important implications for the implementation of automatic proof procedures in higher order logic.

Unification is a fundamental process in all contemporary first-order theorem-provers, since it is embedded as the basic operation in rules such as resolution, factoring and paramodulation (Robinson and Wos, 1969). It is not possible to extend these rules to higher order logic directly, because of the absence of a most general unifier, as shown by Gould (1966). One could of course enumerate the unifiers, but our result shows that we would not know in general when to stop. This is basically the method described in Pietrzykowski and Jensen (1972). Another possible way of automating higher order logic is proposed in Robinson (1969), and this method does not require the process of unification. Unfortunately it would probably suffer the same disadvantages as Herbrand-base saturation methods for first-order logic.

A new method trying to overcome the difficulties of unifier-based methods is presented in Huet (1972). Basically, this procedure computes unifiers for a sequence of cuts leading to a refutation, rather than for an individual cut. The main advantage here is that, by delaying as much as possible the search for unifiers, we save computation time, because the cumulated information permits us to reject a lot of irrelevant cases. Unfortunately we still need to detect the existence of a unifier, and therefore some enumerating process is still necessary.

Acknowledgments

The author wishes to thank Dr. P. Andrews and the referee for their helpful comments and criticisms. The work reported in this paper was supported by the National Science Foundation under Grant No. GJ-1135.

Received: March 9, 1972

REFERENCES

ANDREWS, P. B. (1970), Resolution in type theory, *J. Symbolic Logic* **36**, 414–432.

CHURCH, A. (1940), A formulation of the simple theory of types, *J. Symbolic Logic* **5**, 56–68.

GOULD, W. E. (1966), A matching procedure for ω-order logic. Ph.D. thesis, Princeton University; printed as Scientific Report No. 4, AFCRL 66-781 (Contract AF 19 (628)-3250), AD 646 560.

GUARD, J. R. (1964), Automated logic for semi-automated mathematics. Scientific Report No. 1, AFCRL 64-411 (Contract AF 19 (628)-3250), AD 602 710.

HUET, G. P. (1972), Constrained resolution: a complete method for higher order logic. Ph.D. thesis, Case Western Reserve University; printed as Report 1117, Jennings Computing Center, Case Western Reserve University.

LUCCHESI, C. L. (1972), The undecidability of the unification problem for third order languages. Report CSRR 2059, Department of Applied Analysis and Computer Science, University of Waterloo.

PIETRZYKOWSKI, T. (1971), A complete mechanization of second order logic. Report CSRR 2038, Department of Applied Analysis and Computer Science, University of Waterloo.

PIETRZYKOWSKI, T. AND JENSEN, D. (1972), A complete mechanization of ω-order logic. Report CSRR 2060, Department of Applied Analysis and Computer Science, Univ. of Waterloo. Also, *Assoc. Comp. Mach. Nat. Conf. 1972* **1**, 82–97.

POST, E. L. (1946), A variant of a recursively unsolvable problem, *Bull. Am. Math. Soc.* **52**, 264–268.

ROBINSON, G. A. AND WOS, L. T. (1969), Paramodulation and theorem proving in first-order theories with equality, *Machine Intelligence* **4**, 135–150.

ROBINSON, J. A. (1965), A machine-oriented logic based on the resolution principle, *J. Assoc. Comput. Mach.* **12**, 23–41.

ROBINSON, J. A. (1969), Mechanizing higher order logic, *Machine Intelligence* **4**, 151–170.

ROBINSON, J. A. (1970), A note on mechanizing higher order logic, *Machine Intelligence* **5**, 123–133.