# Unification for Quantified Formulae

by

Wilfried Sieg and Barbara Kauffmann

December 1993

Report CMU-PHIL-44


Philosophy
Methodology
Logic

Pittsburgh, Pennsylvania 15213-3890

# Unification For Quantified Formulae

Wilfried Sieg and Barbara Kauffmann

Department of Philosophy
Carnegie Mellon University
Pittsburgh, PA 15213

ws15@andrew.cmu.edu; tel.: 412-268-8565
bk1k@andrew.cmu.edu; tel.: 412-268-8047

**Abstract.** *Unification* for a first order language is a method that attempts to make terms of the language -- via appropriate substitutions -- syntactically identical. The method can be applied directly to quantifier-free formulae and, in this paper, *will be extended in a natural and straightforward way to quantified formulae.* The main issue is the treatment of bound variables. Our method is based on a canonical renaming of the bound variables occurring in the input formulae, so that every binding is associated in a special way with a distinct variable. Then, unification works as usual by treating logical connectives like functors and bound variables like constants. Our work is meant to help limit the search of theorem proving algorithms, in particular of those that search directly for natural deduction proofs.

**0. Introduction.** Modern automated theorem proving started with work in the late fifties and early sixties that is broadly represented in the volume *Automation of Reasoning* [SW83]. However, the underlying logical techniques went frequently back to pioneering work in the late twenties and early thirties. That work, in particular Herbrand's and Gentzen's, was motivated by foundational concerns connected with Hilbert's Program. Such concerns were replaced by one crucial problem related to the efficient use of Herbrand's fundamental theorem, namely the choice of terms for the instantiation of quantifiers and the identification of contradictory pairs of literals. Herbrand [H71, p. 148] introduced *unification* already for that very purpose, and Robinson [R65] incorporated unification via his *resolution rule* directly into a machine-oriented deductive system. The central question of the unification problem for a first order language **L** is this: Given terms t and t' of **L**, is there a substitution or term-assignment $\sigma$ such that $sub(\sigma,t)$ and $sub(\sigma,t')$ are syntactically identical? (Here $\lambda x.sub(\sigma,x)$ is the unique extension of $\sigma$ to all terms.) A substitution that allows such an identification is called a *unifier* for t and t'. Robinson presented a particular unification algorithm and established that, if the algorithm yields a unifier at all, then it is a *most general unifier* (or mgu).

Unification is not only crucial for resolution, but in fact for any refutation method -- whether based on the *sequent calculus*, *semantic tableaux*, or *mating*, to name just some possibilities (compare [GMW79], [F90], [A81].) We use unification for automated proof search, i.e. in algorithms that search directly for natural deduction proofs in classical first order logic (see [SS89], [SS92], [K93]). The basic tenet is that *searching* for a proof of a *proposition* from given *assumptions* should not necessarily resort to the atomic level. Thus, we have to solve the unification problem simultaneously for terms and quantified formulae, or for *q-terms*; if we denote by "quantified terms" the general category of syntactic expressions including terms as well as formulae. As the main technical device for our solution we introduce the three place *C-substitution operation* $sub_C$ and formulate the generalized problem for q-terms e and e' as follows:

- Is there a substitution $\sigma$ such that $sub_C(\sigma,id,e) = sub_C(\sigma,id,e')$?

That means, informally, does $\sigma$ make e and e' identical modulo renaming of bound variables? -- The spirit of our solution (though developed independently) is in line with Snyder's general strategy in [S91, p. 4]: by examining what it should mean for two expressions to be "the same" after a unifying substitution, the appropriate extensions or modifications of the (Herbrand-style) unification method suggest themselves naturally. Our solution is clearly not tied to the particular motivation from automated proof search, but can be used in theorem proving algorithms based on refutation methods or, for that matter, in quite different contexts, where unification is a useful tool; see [KK89]. Finally, the simplicity of our approach should be compared to the complexity of the proposal made by Staples and Robinson in [SR88] and [SR90].

**1. C-substitution.** The *first order* languages **L** we use consist of the following symbols: *universal* and *existential* quantifiers: $\forall$, respectively $\exists$; variables $x_n$, $n \in N$; sentential connectives including $\neg$, $\wedge$, $\vee$, $\rightarrow$; for each $n \in N$, countably many n-place relation symbols and n-ary function symbols. Terms t, t', ... and formulae A, A', ... are inductively defined as usual, but taken to be in prefix notation; *q-terms* e, e', ... are either terms or formulae. It can be shown that q-terms have a unique presentation $ue_1 \ldots e_n$, where u is a variable (with n=0), a logical connective, a relation or function symbol of the appropriate (number of) arguments. Thus, we can give recursive definitions and inductive arguments for q-terms considering three cases: e is a variable, e is of the form $@e_1 \ldots e_n$ (where @ is a sentential connective, a relation or function symbol), or e is of the form $Qx_ne'$ (where Q is either the universal or existential quantifier). The following expressions are examples of q-terms: the atomic formula $Pt_1 \ldots t_n$, the conditional $\rightarrow\wedge AA'\vee AA'$, and the quantified statement $Qx_nA$. The set of free variables of a q-term e is denoted by FV(e).

A *term-assignment* $\sigma$ is a function from the set V of all variables to the set of terms, such that the set of variables $x_n$ for which $\sigma(x_n)\neq x_n$ is finite; this finite subset of V is called the *support* of $\sigma$, briefly $\sup(\sigma)$. The term-assignment with empty support is obviously the *identity function* id on V. The set of variables $FV(\sigma)$ of a term-assignment $\sigma$ is the set of variables occurring in the terms $\sigma(x_n)$ for $x_n \in \sup(\sigma)$. A term-assignment for variables can be uniquely extended to a substitution operation on all terms. We will use $\sigma$ to indicate a term-assignment or its uniquely associated extension; indeed, following common usage, we will call term-assignments also substitutions. Given substitutions $\sigma$ and $\eta$, their *composition* $\sigma\eta$ is defined by $\sigma\eta(x_n)=\sigma(\eta(x_n))$ for all variables $x_n$. A substitution $\sigma$ is called *idempotent* iff $\sigma\sigma = \sigma$.

To extend substitutions not only to terms, but to q-terms, we use modified term-assignments $\sigma^{t}_{x_n}$; the latter have value $\sigma(x_m)$ for variables different from $x_n$, and they have value t for $x_n$. The *replacement operation* rep(σ,e) is defined by:

1. $rep(\sigma,x_n) = \sigma(x_n)$;
2. $rep(\sigma,@e_1 \ldots e_n) = @rep(\sigma,e_1) \ldots rep(\sigma,e_n)$;
3. $rep(\sigma,Qx_n\ e) = Qx_n\ rep(\sigma^{x_n}_{x_n},\ e)$.

This operation replaces all free occurrences of a variable in a q-term e by its σ-value, but leaves bound occurrences untouched. From a logical point of view, however, rep is an incorrect "substitution operation" for q-terms e involving quantifiers: it allows the replacement of free variables in the scope of a quantifier by terms that contain the quantified variable. We have two well-known choices: either we require terms $\sigma(x_n)$ to be free for $x_n$ in e, or we rename bound variables in such a way that this requirement is always satisfied. We take the second alternative and define the *substitution operation* sub(σ,e) with *minimal and canonical renaming* of bound variables. Let $\sigma|_e$ be "σ restricted to e" so that $\sigma|_e(x_n) = \sigma(x_n)$, if $x_n \in FV(e)$, and $\sigma|_e(x_n)=x_n$ otherwise. sub(σ,e) is defined as follows:

1. $sub(\sigma,x_n) = \sigma(x_n)$;
2. $sub(\sigma,@e_1 \ldots e_n) = @sub(\sigma,e_1) \ldots sub(\sigma,e_n)$;
3. $sub(\sigma,Qx_ne) = \begin{cases} Qx_n\ sub(\sigma^{x_n}_{x_n},e) & \text{if } x_n \notin FV(\sigma|_e) \text{ or } x_n \notin FV(e) \\ Qx_m\ sub(\sigma^{x_m}_{x_n},e) & \text{otherwise} \end{cases}$

where $x_m$ is the first variable that does not belong to $FV(\sigma|_e) \cup FV(e)$.
Considering id for σ we have the operation sub(id,e) that renames the bound variables of e. Finally, the *C-substitution operation* $sub_C(\sigma,\tau,e)$ performs a substitution and generates a *canonical renaming* relative to a context given by a set C of variables.

1. $sub_C(\sigma,\tau,x_n) = \begin{cases} \tau(x_n) & \text{if } x_n \in sup(\tau) \\ \sigma(x_n) & \text{otherwise} \end{cases}$
2. $sub_C(\sigma,\tau,@e_1 \ldots e_n) = @sub_C(\sigma,\tau,e_1) \ldots sub_C(\sigma,\tau,e_n)$
3. $sub_C(\sigma,\tau,Qx_ne) = Qx_m sub_C(\sigma,\tau^{x_m}_{x_n},e)$, where $x_m$ is the first variable whose subscript is greater than the subscripts of all variables in $C \cup FV(e) \cup FV(\sigma) \cup FV(\tau)$.

$sub_C(id,id,e)$ yields a canonical renaming of bound variables in e relative to the context C. That this renaming has the desired logical effect is shown by the next lemma.

**Lemma 1.1.** Let e be a q-term and $\sigma$, $\tau$ substitutions, and assume that C is a finite set of variables such that (i) $sup(\sigma)\cup FV(e)\cup FV(\sigma)$ is included in C and (ii) the subscripts of variables in $FV(\tau)$ are all greater than those of elements in C; then we have:

$$sub_C(\sigma,\tau,e)=rep(\sigma,sub_C(id,\tau,e)).$$

**Proof.** The proof proceeds by induction on q-terms e; the case that e is a variable follows directly from the definitions, the case that e is of the form $@e_1 \ldots e_n$ from induction hypothesis. The case that $e=Qx_ne_1$ is the critical one; here we have by definition

(1) $sub_C(\sigma,\tau,Qx_ne_1)=Qx_msub_C(\sigma,\tau^{x_m}_{x_n},e_1)$,

where $x_m$ is the first variable with subscript greater than the subscripts of variables in $C\cup FV(\tau)$; by induction hypothesis we have:

(2) $Qx_msub_C(\sigma,\tau^{x_m}_{x_n},e_1)=Qx_mrep(\sigma,sub_C(id,\tau^{x_m}_{x_n},e_1))$.

As $x_m$ is not in $sup(\sigma)$, $\sigma=\sigma^{x_m}_{x_m}$ and thus by definition of rep:

(3) $Qx_mrep(\sigma,sub_C(id,\tau^{x_m}_{x_n},e_1))=rep(\sigma,Qx_msub_C(id,\tau^{x_m}_{x_n},e_1))$.

By choice of m and definition of $sub_C$:

(4) $rep(\sigma,Qx_msub_C(id,\tau^{x_m}_{x_n},e_1))=rep(\sigma,sub_C(id,\tau,Qx_ne_1))$.

The claim follows from (1)-(4). **Q.E.D.**

The lemma insures that $sub_C$ behaves on a q-term e as replacement does on the canonically renamed q-terms $e^*=sub_C(id,id,e)$, because $sub_C(\sigma,id,e)=rep(\sigma,e^*)$. -- The operation $sub_C$ will be used for the unification of q-terms e and e', where the context C is provided by the set $FV(e)\cup FV(e')$. Indeed, we call e and e' *unifiable* if and only if there is a term-assignment $\sigma$ with $sub_C(\sigma,id,e)=sub_C(\sigma,id,e')$ and $C=FV(e)\cup FV(e')$; the substitution $\sigma$ is called a unifier for e and e'.[1] In section 3 we present a unification algorithm for q-terms, whose output determines a unifier $\sigma$ for e and e', indeed an mgu. Where a unifier $\sigma$ for e and e' is a most general unifier for e and e' if $\sigma$ is more general than any other unifier $\eta$ for those q-terms; i.e., for some substitution $\tau$, we have $\eta=\tau\sigma$. But first we should convince ourselves that $sub_C(\sigma,\tau,e)$ is an operation providing an appropriate identity criterion for the "unifiability of q-terms" by considering a few (typical) examples.

[1] The role of the second argument is similar to that of the "binding states" in [SR88] and [SR90]: the substitution $\tau$ keeps track of information on bound variables.

To re-emphasize, we want to find a unification algorithm mainly for automated proof search and use unification extensively, in order to be able to fill the gap between goal and premises without necessarily resorting to the atomic level. Let us make this point clearer with an example. Suppose we want to derive the goal $G=\vee\forall x_n P x_n f g x_m g u R x_n x_m$ (where the free variables in G are taken to be existentially quantified) from a set of propositions S containing $A=\vee\forall x_n P x_n f x_j g x_m R g u x_m$ (where the free variables of A are taken to be universally quantified). We would like to recognize the provability of G from S *in one step* without decomposing G and A, since a substitution $\sigma$ exists, namely $\{\langle x_j, g x_m\rangle, \langle x_m, u\rangle, \langle x_n, g u\rangle\}$, such that $sub_C(\sigma,id,G)=sub_C(\sigma,id,A)$. Notice, that the values of $\lambda x.sub_C(\sigma,id,x)$ for A and G are syntactically identical expressions; i.e., we consider formulae to be unifiable, if the application of $sub_C$ for a suitable term-assignment yields syntactic identities! That corresponds perfectly to the ordinary understanding of bound variables as refelected graphically in Bourbaki's way of indicating bindings. Based on this understanding, the following examples show when a unification algorithm for q-terms should succeed and when, instead, it should fail:

1. $Qx_nPx_n$ and $Qx_mPx_m$ should unify.
2. $Qx_nPx_n$ and $Qx_mPa$ should not unify.
3. $Qx_nPx_m$ and $Qx_mPx_n$ should unify, but not with $Qx_kPx_k$.
4. $Qx_nQx_nPx_nx_n$ and $Qx_nQx_mPx_nx_m$ should not unify.
5. $Qx_nQx_mPx_nx_m$ and $Qx_mQx_nPx_nx_m$ should not unify.

Note that in the last case unification should fail even though the two formulas are logically equivalent, as our approach to quantificational unification is not modulo quantification theory. It will be interesting to explore the efficiency of proof search procedures, when invoking unification algorithms with built-in logical identities (like the one in 5).

**2. An equational calculus.** The work in this section is preliminary to that in section 3, where we present Martelli and Montanari's unification algorithm [MM 82] extended to q-terms. In rough outline the algorithm proceeds as follows. Given e and e', it constructs sets of equations, starting from the equation between the renamed q-terms $sub_C(id,id,e)$ and $sub_C(id,id,e')$ with $C=FV(e)\cup FV(e')$ and then transforming sets according to the rules formulated below, until it fails or until it cannot apply any transformation rule. In the latter case the output will be a system of equations that determines a unifier.

We introduce four transformations on systems of equations. Note that the distinction between bound and free variables (of the input formulae) can be made by looking at their

subscripts, since the q-terms we are considering have been renamed. Bound variables are of the form $x_n$ with $n \geq k$, where k is the successor of the largest number used as a subscript of a free variable occurring in the input q-terms; while *free* variables are of the form $x_n$ with $n<k$. (As above, $e_i=e_j$ expresses that the q-terms $e_i$ and $e_j$ are syntactically identical, whereas $e_i \approx e_j$ denotes an element of a system E of equations.)

(1) Exchange.

$\{e \approx x_n\} \cup E \Rightarrow \{x_n \approx e\} \cup E$, if e is not a variable.

(2) Equation Elimination.

$\{x_n \approx x_n\} \cup E \Rightarrow E$.

(3) Q-Term Reductions.

$\{@e_1 \ldots e_n \approx @e'_1 \ldots e'_n\} \cup E \Rightarrow \{e_1 \approx e'_1, \ldots, e_n \approx e'_n\} \cup E$;[2]

$\{Qx_n e_1 \approx Qx_n e_2\} \cup E \Rightarrow \{e_1 \approx e_2\} \cup E$.

(4) Variable Elimination.

$\{x_n \approx t\} \cup E \Rightarrow \{x_n \approx t\} \cup \{\langle x_n, t\rangle\}E$[3], provided $n<k$ and $x_n$ does not occur in t; furthermore, t must not contain any variable $x_m$ with $m \geq k$.

We want to show that the application of these transformation rules to the singleton $E=\{sub_C(id,id,e) \approx sub_C(id,id,e')\}$ leads to a system of equations that determines a unifier for e and e' (if e and e' are indeed unifiable). If we can find a term-assignment $\sigma$ such that $rep(\sigma,sub_C(id,id,e))=rep(\sigma,sub_C(id,id,e'))$, then we are done, as by Lemma 1.1 the latter equation holds if and only if $sub_C(\sigma,id,e)=sub_C(\sigma,id,e')$, i.e., e and e' are unifiable. -- We call a set of equations E* *derived from E* iff it has been obtained from E by successive applications of the rules (1) through (4). Finally, $\sigma$ is called a *solution for E* iff $rep(\sigma,e_j)=rep(\sigma,e_k)$ for every equation $e_j \approx e_k$ in E; E and E' are *equivalent* iff E and E' have the same set of solutions. Now we can formulate and prove the following theorem.

**Theorem 2.1.** Any system E* of equations derived from an equation E between canonically renamed q-terms is equivalent to E.

**Proof** (by induction on the number of rule applications). Exchange and equation elimination preserve obviously equivalence to E. Let us consider the remaining rules.

---

[2] Note that in case n=0, the set of new equations is clearly empty.

[3] This indicates replacement of t for $x_n$ throughout E.

Assume E* has been obtained from a derived set of equations E' by a q-term reduction; this transformation replaces equations of the form $@e_1 \ldots e_n \approx @e'_1 \ldots e'_n$ in E' by equations $e_1 \approx e'_1, \ldots, e_n \approx e'_n$ and those of the form $Qx_n e_1 \approx Qx_n e_2$ by $e_1 \approx e_2$. But any solution $\sigma$ for E' is a solution for E* and vice versa, as $rep(\sigma,e_i)=rep(\sigma,e'_i)$ for all i iff $rep(\sigma,@e_1 \ldots e_n)=rep(\sigma,@e'_1 \ldots e'_n)$, respectively $rep(\sigma,Qx_ne_1)=rep(\sigma,Qx_ne_2)$, recalling the proof of Lemma 1.1. Therefore, E* and E' are equivalent; but by induction hypothesis E' is equivalent to E, thus E* is equivalent to E.

Assume now that E* has been obtained from $E'=E'' \cup \{x_n \approx t\}$ by variable elimination, where $x_n$ is a free variable (i.e., $n<k$) and does not occur in t; furthermore, t must not contain any variable $x_m$ with $m \geq k$. This transformation replaces $x_n$ by t in all equations of E'' and yields E* by joining $\{x_n \approx t\}$ to $\{\langle x_n,t\rangle\}E''$. Solutions for E' are clearly also solutions for E*. So let $\sigma$ be a solution for E*; as E* contains the equation $x_n \approx t$, $\sigma$ is a solution for it and also for all equations in E'' not containing $x_n$, as the latter equations occur unchanged in E*. For equations $u \approx v$ of E'' containing $x_n$, E* will have corresponding equations $u' \approx v'$ with $\{\langle x_n,t\rangle\}(u)=u'$ and $\{\langle x_n,t\rangle\}(v)=v'$. But then $rep(\sigma,u)=rep(\sigma,u')$ and $rep(\sigma,v)=rep(\sigma,v')$. Thus, every solution for E* is also a solution for E' (and thus E* is, as above, equivalent to E). **Q.E.D.**

A system E* of equations is in *solved form* iff the following two conditions are satisfied: (a) every equation is of the form $x_n \approx t$, where n is less than k and the subscripts j of all variables occurring in t are less than k and different from n; (b) if $x_n$ is the left member of some equation, then it does not occur in any other equation of E*. The equations of such an E* obviously determine a term-assignment $\sigma$ by setting $\sigma(x_n)=x_n$, if $x_n$ does not occur on the left-hand side of any equation in E*, and $\sigma(x_n)=t$, if $x_n \approx t$ is in E*. If E* is in solved form (derived from $E=\{sub_C(id,id,e) \approx sub_C(id,id,e')\}$ for q-terms e and e') the $\sigma$ just defined is actually an idempotent mgu for the input q-terms. After all, $\sigma$ is a solution for E* and, by Theorem 2.1, of E; thus $\sigma$ is a unifier for e and e'. Because of condition (b) for systems of equations in solved form, $\sigma$ must be idempotent and, because of (a), an mgu. So we have:

**Theorem 2.2.** Let E* be derived from $E=\{sub_C(id,id,e) \approx sub_C(id,id,e')\}$ for arbitrary q-terms e and e'; if E* is in solved form, then it determines a solution $\sigma$ that is an idempotent mgu for e and e'.

Having established that a derived set E* of equations in solved form determines a unifier, we formulate now an algorithm that decides the unifiability of two q-terms and provides an

idempotent mgu in case the decision is positive. Indeed, the algorithm $\mathfrak{U}$ will transform the initial equation between the canonically renamed q-terms systematically into a system of equations in solved form, if the q-terms are unifiable; if they are not unifiable, the algorithm will end with "failure".

**3. The unification algorithm $\mathfrak{U}$.** The algorithm $\mathfrak{U}$ constructs a sequence of systems of equations starting from $\{sub_C(id,id,e) \approx sub_C(id,id,e')\}$ by applying the following operations:

1. select an equation of the form $t \approx x_n$ and apply *exchange*;
2. select an equation $x_n \approx t$ ,
   a. if $t = x_n$, apply *equation elimination*;
   b. if $t \neq x_n$
      i. if $x_n$ is a bound variable, i.e., $n \geq k$, then *fail*;
      ii. if $x_n$ or a bound variable occurs in t, then *fail*;
      iii. if $x_n$ occurs in some other equation, apply *variable elimination*;
3. select an equation of the form $e \approx e'$.
   a. If the root symbols (function, predicate, connective or quantifier symbols[4]) or their arities are different, then *fail*;
   b. otherwise, apply *term reduction*;
4. If none of the previous operations can be applied, then *stop* .

An *example* will highlight the crucial features of $\mathfrak{U}$. Consider q-terms $e=\forall x_5 P c h x_5 x_3$ and $e'=\forall x_8 P x_5 h x_8 x_5$. A canonical renaming of all bound variables leads to q-terms $e^*=\forall x_9 P c h x_9 x_3$ and $e'^*=\forall x_9 P x_5 h x_9 x_5$ and the initial set of equations $\{e^* \approx e'^*\}$. $\mathfrak{U}$ applies *term reduction* to get $\{P c h x_9 x_3 \approx P x_5 h x_9 x_5\}$. Both terms start with the same predicate symbol P, so $\mathfrak{U}$ applies term reduction again to obtain $\{c \approx x_5, h x_9 x_3 \approx h x_9 x_5\}$ and uses *exchange* to rewrite the first equation as $x_5 \approx c$. *Variable elimination* with $x_5 \approx c$ applied to $\{h x_9 x_3 \approx h x_9 x_5, x_5 \approx c\}$ and subsequent term reduction for $h x_9 x_3 \approx h x_9 c$ yields the set $\{x_9 \approx x_9, x_3 \approx c, x_5 \approx c\}$. $\mathfrak{U}$ drops the first equation from the set by *equation elimination* to obtain $E^*=\{x_3 \approx c, x_5 \approx c\}$. No other rule can be applied, so $\mathfrak{U}$ will stop with the set of equations $E^*$. Note that $E^*$ is in solved form.

[4] We consider a quantifier symbol Q together with the first occurrence of the variable $x_n$ bound by Q, i.e., $Qx_n$.

**Theorem 3.1** (Termination and correctness of $\mathfrak{U}$). Let E be the system of equations $\{sub_C(id,id,e) \approx sub_C(id,id,e')\}$ for arbitrary q-terms e and e'; $\mathfrak{U}$ applied to E always terminates: **if** $\mathfrak{U}$ terminates with failure, then e and e' are not unifiable; otherwise, $\mathfrak{U}$ terminates with step 4 and presents a system $E^*$ of equations in solved form that is equivalent to E.

**Proof.** $\mathfrak{U}$ applies rules of the equational calculus, unless $\mathfrak{U}$ meets one of the exit conditions. If $\mathfrak{U}$ exits through one of the failure points 2.b.i, 2.b.ii, or 3.a, then $\mathfrak{U}$ terminates and the terms e and e' are obviously not unifiable. Otherwise only steps 1., 2.a, 2.b.iii, or 3.b are applied. We claim that in this case E is eventually transformed into a system $E^*$, such that none of these steps is applicable, and $\mathfrak{U}$ terminates on account of 4. As E* is derived from E, E* and E are equivalent by Theorem 2.1. And because of the termination condition, E* must be in solved form. That establishes the correctness of $\mathfrak{U}$ also in this case -- if $\mathfrak{U}$ indeed terminates.

To establish termination of $\mathfrak{U}$ under the stated circumstances, it suffices to show that any sequence of steps 1., 2.a, 2.b.iii, or 3.b must be finite, when started with an application to E. For that purpose we associate with any set E' of equations a triple of natural numbers $cp(E')=\langle n_1,n_2,n_3\rangle$, where $n_1$ is the number of free variables with more than one occurrence, $n_2$ is the number of occurrences of function symbols, relation symbols, and of logical connectives (i.e., quantifiers as well as sentential connectives), and $n_3$ is the number of equations that are either of the form $x_n \approx x_n$ or of the form $t \approx x_n$ with $x_n$ a free variable and t a non-variable term.

The lexicographic ordering $>$ of triples of natural numbers is a well-ordering; thus, the finiteness claim has been established, as soon as we have shown that any single application of steps 1., 2.a, 2.b.iii, or 3.b to an equation E' leads to an equation E'' with $cp(E') > cp(E'')$. Recall that $\langle n_1,n_2,n_3\rangle > \langle n'_1,n'_2,n'_3\rangle$ holds if at least one $n_i > n'_i$, for $1 \le i \le 3$, and all $n_j = n'_j$ for $0 \le j < i \le 3$. Now consider the various possible steps applicable to E':

*step 1.* of $\mathfrak{U}$, exchange, reduces only the number $n_3$ of equations of the form $t \approx x_n$, leaving all the other factors unchanged, so $cp(E') > cp(E'')$;

*step 2.a* of $\mathfrak{U}$, equation elimination, reduces the number $n_3$ of equations of the form $x_n \approx x_n$, may reduce $n_1$, but does not effect $n_2$, thus we have in any case $cp(E') > cp(E'')$;

2.b.iii of $\mathfrak{U}$, variable elimination, reduces $n_1$ and may increase $n_2$, but even if $n_2$ is increased, we have that $cp(E') > cp(E'')$;

3.b of $\mathfrak{U}$, term reduction, reduces $n_2$ and may increase $n_3$, but even if $n_3$ is increased, we have that $cp(E') > cp(E'')$. **Q.E.D.**

**4. Final remarks.** First, Martelli and Montanari's algorithm has been modified only to treat bound variables. It seems to be clear that other algorithms, e.g. from [PW78], can be modified in a similar way; our way of proceeding provides a case study. The complexity of the original algorithm is not essentially increased. Indeed, even the addition of canonical renaming does not change the overall complexity, as that procedure is only linear in the input.

Second, our goal was to limit the search for theorem provers, in particular for those used in automated proof search. From that, the need of an algorithm for the unification of quantified terms arose. Our proposal to use a canonical renaming seems to be a very natural way of treating bound variables in this context. Indeed, after renaming, our unification algorithm behaves as a usual unification algorithm for terms, treating quantifiers as functors and bound variables as special constants.

**Acknowledgments.** We would like to thank Alberto Momigliano and Frank Pfenning for interesting suggestions and fruitful conversations.

## References.

[A81] P. Andrews, Theorem Proving via General Matings, JACM 28 (2), pp.193-214.

[F90] M.Fitting, *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, 1990.

[GMW79] M.J.C.Gordon, R.Milner and C.P.Wadsworth, *Edinburgh LCF: A Mechanised Logic of Computation.* Springer LNCS 78, 1979.

[H71] J. Herbrand, *Logical Writings*, W. Goldfarb (ed.), Harvard University Press, Cambridge 1971.

[K93] B.Kauffmann, *A Unification Algorithm for Quantified-Formulae*, Master Thesis, Department of Philosophy, Carnegie Mellon University, May 1993.

[KK89] K.Knight, Unification: a Multidisciplinary Survey, *ACM Computing Surveys*, Vol.21, No.1, pp.93-124, 1989.

[MM76] A.Martelli and U.Montanari, Unification In Linear Time and Space: A Structured Presentation, *Istituto Di Elaborazione Dell'Informazione (Pisa)*, B76-16, 1976.

[MM82] A.Martelli and U.Montanari, An Efficient Unification Algorithm, *ACM Transaction on Programming Languages and Systems*, Vol.4, No.2, pp.258-282, 1982.

[PW78] M.S.Paterson and M.N.Wegman, Linear unification, *Journal of Computer and System Sciences*, Vol.16, N0.2, pp.158-167, 1978.

[R65] J.A.Robinson, A Machine-oriented Logic Based on the Resolution Principle, *JACM*, Vol.12, No.1, pp.23-41, 1965.

[SS89] W.Sieg and R.Scheines, Automated Proof Search (in sentential logic), in: *Philosophy and the Computer*, L. Burkholder (ed.), Westview Press, 1992, pp. 137-159. [The paper was presented to Conference on Computing in Philosophy, Pittsburgh, 1989.]

[SS92] W.Sieg and R.Scheines, Searching for Proofs (in predicate logic), manuscript, December 1992.

[SW83] J. Siekmann and G. Wrightson, *Automation of Reasoning* (Classical Papers on Computational Logic 1957-1966), vol. 1, Springer-Verlag, 1983.

[S91] W.Snyder, *A Proof Theory For General Unification*, Birkhäuser, Boston, Basel, Berlin, 1991.

[SR88] J.Staples and P.J.Robinson, Efficient Unification Of Quantified Terms, *The Journal of Logic Programming* , No.5, pp.133-149, 1988.

[SR90] J.Staples and P.J.Robinson, Structure Sharing for Quantified Terms: Fundamentals, *Journal of Automated Reasoning* , No.6, pp.115-145, 1990.