Analysis of Fast Unification: An Exercise in 
Applied Logic  
Kaj Børge Hansen

1 Preliminaries

1.1 Introduction. In this essay, I continue the investigation of the unification process and of algorithms for this operation which was begun in Hansen (1996). In Section 2, I first prove that the unification algorithm in § 1.4 does exactly what we want it to do. Then I show that Algorithm 1.4 is of exponential complexity and therefore really unsuited for electronic computers. The rapid growth of complexity is due to the presence of the occur check. This is the reason why, in practical implementations of the unification algorithm, the occur check normally is omitted. This omission can lead to unsoundness in PROLOG. It turns out that these inconsistencies in PROLOG to some extent can be isolated and do not lead to a collapse of the system. Thus the Shepherdson-Tärnlund theorem can still be proved with the occur check omitted. In other words, PROLOG still makes correct computations even in implementations where the occur check is absent from the unification algorithm.

Martelli and Montanari (1982) have developed an alternative, sound unification algorithm, including the occur check, which is of only polynomial complexity. It is therefore computationally much more efficient than the algorithm 1.4 and better suited for electronic computers. In the second part of Section 2, I give an exposition of the Martelli-Montanari algorithm and prove that it is adequate and polynomial. Conceptually, this algorithm is somewhat more difficult and involved than algorithm 1.4; but computationally, it is superior.

1.2 Definition. (I) A substitution set

\[ \sigma = \{ x_1 = t_1, \ldots, x_n = t_n \} \]

is a set of identity formulas

\[ x_i = t_i \]
where $x_i$ is a variable and $t_i$ is a term. It is assumed that no variable occurs as left-hand side in two distinct identities in a substitution set. Thus we write a substitution set as a set of equations.

(II) If $\sigma$ is a substitution set and $U$ is an expression, then $U\sigma$ is the result of substituting $t_i$ for all occurrences of $x_i$ in $U$ ($i = 1, \ldots, n$).

1.3 Rule for Compositions. Let

$$\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$$
$$\theta = \{y_1 = u_1, \ldots, y_m = u_m\}$$

be substitutions sets. Then the composition of them is the substitution set

$$\sigma\theta = \{x_1 = t_1\theta, \ldots, x_n = t_n\theta, y_1 = u_1, \ldots, y_m = u_m\}$$
$$\text{- } \{x_i = t_i\theta \mid t_i\theta \text{ is } x_i, 1 \leq i \leq n\}$$
$$\text{- } \{y_j = u_j \mid y_j \in \{x_1, \ldots, x_n\}, 1 \leq j \leq m\}$$

1.4 The Unification Algorithm (Robinson). I give a formulation of Robinson's unification algorithm which determines a Most General Unifier (MGU) for two terms or for two atomic formulas. The formulation is a slightly modified and corrected version of the algorithm as given in Genesereth and Nilsson (1987).

Recursive Procedure $\text{Mgu}(u,v)$

Begin

$u = v \Rightarrow \text{Return } \{\}$,
Variable($u$) $\Rightarrow \text{Return } \text{Mguvar}(u, v)$,
Variable($v$) $\Rightarrow \text{Return } \text{Mguvar}(v, u)$,
Constant($u$) or Constant($v$) $\Rightarrow \text{Return } \text{False}$,
Not (Length($u$) = Length($v$)) $\Rightarrow \text{Return } \text{False}$,
Not (Part($u$, 0) = Part($v$, 0)) $\Rightarrow \text{Return } \text{False}$,
Begin $i := 1$,
$\sigma := \{\}$,
Tag $i > \text{Length}(u) \Rightarrow \text{Return } \sigma$,
$\theta := \text{Mgu}(\text{Part}(u, i), \text{Part}(v, i))$,
$\theta = \text{False} \Rightarrow \text{Return } \text{False}$,
$\sigma := \text{Compose}(\sigma, \theta)$,
$u := \text{Substitute}(u, \sigma)$,
$v := \text{Substitute}(v, \sigma)$,
$i := i+1$,
$\text{Goto Tag}$
End
End
Analysis of Fast Unification

Procedure \texttt{Mguvar}(u,v)

\begin{flushleft}
\textbf{Begin}
\begin{itemize}
  \item Not Term(v) \implies \texttt{Return False},
  \item Occur(u,v) \implies \texttt{Return False},
  \item \texttt{Return} \{u = v\}
\end{itemize}
\textbf{End}
\end{flushleft}

If the two expressions $u$ and $v$ are unifiable, the procedure gives an MGU. If $u$ and $v$ are not unifiable, the algorithm answers \texttt{False}. There are several undefined components in the procedure:

Variable(u) is true $\iff$ u is a variable
Constant(u) is true $\iff$ u is a constant
Term(u) is true $\iff$ u is a term
Occur(u, v) is true $\iff$ u is a variable which occurs in the term v
Length($P(t_1, ..., t_n)$) = the number of places in $P = n$
Length($f(t_1, ..., t_n)$) = the number of places in $f = n$
Part($P(t_1, ..., t_n)$, 0) = $P$
Part($P(t_1, ..., t_n)$, i) = $t_i$ for $1 \leq i \leq n$
Part($f(t_1, ..., t_n)$, 0) = $f$
Part($f(t_1, ..., t_n)$, i) = $t_i$ for $1 \leq i \leq n$
Compose($\sigma$, $\theta$) = the composition $\sigma \theta$ of the substitutions $\sigma$ and $\theta$
Substitute($u$, $\sigma$) = the result $u \sigma$ of applying the substitution $\sigma$ to the expression $u$.

1.5 \textbf{Theorem} (Shepherdson-Tärlund). Every recursive function can be computed in Horn clause logic and in PROLOG.

\textit{Proof} (Hansen (1996)):
The proof is by induction on the definition of recursive functions. We operate in the language $L(PA) = \{0, S, +, \cdot, <\}$ of Peano arithmetic.

\textbf{(RF1)}: Define the zero-function $Z$ by
\begin{equation}
Z(x) = 0 \leftarrow (1-1)
\end{equation}

Since in $L(PA)$ the number $n$ is represented by $SS...S0$ where 'S' occurs $n$ times in the sequence, the equation
\begin{equation}
S(x) = x + 1
\end{equation}
is an instance of the identity axiom
\begin{equation}
x = x \leftarrow (1-2)
\end{equation}
If we represent $S$ by the 2-place predicate $\delta$, then $S$ is defined by the possibly more satisfactory clause

$$\delta(x, S(x)) \leftarrow$$

The projection function $U_{n,i}$ is defined by

$$U_{n,i}(x_1, \ldots, x_i, \ldots, x_n) = x_i \leftarrow$$

**RF2:** Let $f$ be defined by composition from $g$, $h_1$, ..., $h_n$. By the induction hypothesis, there are Horn clauses which define $g$, $h_1$, ..., $h_n$. Define

$$f(x) = y \leftarrow h_1(x) = y_1, \ldots, h_n(x) = y_n, g(y_1, \ldots, y_n) = y$$

**RF3:** By the induction hypothesis, there are Horn clause procedures for computing $h$ and $g$. Then $f$ is defined by the Horn clause procedure

(1-5) $f(x,0) = z \leftarrow z = g(x)$

(1-6) $f(x,y+1) = z \leftarrow z = h(x,y,f(x,y))$

We have now shown that all primitive recursive functions are Horn clause computable. All primitive recursive functions can therefore be used in the computations in Horn clause logic of recursive functions defined by minimisation.

**RF4:** Let $f$ be defined by

$$f(x) = \mu y (g(x,y) = 0)$$

where $g$ is recursive and satisfies $\forall x \exists y g(x,y) = 0$. By the induction hypothesis, $g$ can be computed in Horn clause logic. We define a function $P_g(x,y)$ by

(1-7) $P_g(x,0) = z \leftarrow z = \text{Sign}(g(x,0))$

(1-8) $P_g(x,y+1) = z \leftarrow z = P_g(x,y) \cdot \text{Sign}(g(x,S(y)))$

where $\text{Sign}(0) = 0$ and $\text{Sign}(z+1) = 1$ for all $z$. $P_g$ is defined using (RF3), primitive recursion. $g$ is computable in Horn clause logic by the induction hypothesis. $\text{Sign}$, multiplication, and the successor function $S$ are primitive recursive and therefore computable in Horn clause logic by the first part of the present proof. It follows that $P_g$ is computable in Horn clause logic. $P_g$ is the function

$$P_g(x,y) = \text{Sign}(g(x,0)) \cdot \ldots \cdot \text{Sign}(g(x,y))$$

$P_g(x,y)$ has the following nice properties:
(1-9) If $P_g(x,0) = 0$, then $\mu y \ (g(x,y) = 0) = 0$.
If $P_g(x,0) > 0$, then $P_g(x,y)$ changes value exactly once, namely the first time $g(x,y) = 0$.

From (1-9) and the definition of $P_g$ follows
(1-10) If $P_g(x,0) > 0$ and $P_g(x,a) \neq P_g(x,a+1)$, then $\mu y \ (g(x,y) = 0) = a+1$.

From (1-9) and (1-10) we see that $f$ can be defined uniquely by the Horn clauses
(1-11) $f(x) = 0 \leftarrow P_g(x,0) = 0$
(1-12) $f(x) = y+1 \leftarrow P_g(x,y) = 1, \ P_g(x,y+1) = 0$

Since $P_g$ is Horn clause computable, the Horn clauses (1-11) and (1-12) show that $f$ is computable in Horn clause logic and in PROLOG.

2 The Martelli-Montanari Unification Algorithm

2.1 Introduction. In this section I first prove that the Robinson unification algorithm does what we want it to do. Next I prove that it is of exponential complexity and that the omission of the occur check from the unification algorithm does not make PROLOG computationally unsound. Most of the section is devoted to a detailed development of the polynomial Martelli-Montanari unification algorithm.

2.2 Theorem (Robinson). Let $U$ and $V$ be atomic formulas or terms fed into the unification algorithm 1.4. Then:
(2-1) The algorithm halts after a finite number of steps.
(2-2) If the algorithm answers False, then $U$ and $V$ are not unifiable.
(2-3) If the algorithm answers $\sigma = \sigma_1 \ldots \sigma_n$, then $U$ and $V$ are unifiable and $\sigma$ is a unifier.
(2-4) For every unifier $\theta$ for $U$ and $V$, we have $\theta = \sigma \ \theta$ which implies that $\sigma$ is a MGU for $U$ and $V$.

Proof:
In each step, the algorithm performs one of the following operations:
(i) it returns False, or
(ii) it returns $\sigma$, or
(iii) it makes a substitution \( \{ x = t \} \) where \( x \) is a variable occurring in \( U \) or \( V \) and \( t \) is a term which does not contain \( x \) and all of whose symbols occur in \( U \) or \( V \).

Each of the steps (i) and (ii) terminates the algorithm. Each step (iii) lowers the number of variables in \( U_\sigma \) and \( V_\sigma \) by one variable. At the latest when no variable is left in any of the two expressions \( U_\sigma \) and \( V_\sigma \), the algorithm must return \textbf{False} or \( \sigma \) after which step it halts.

By inspecting the algorithm, we see that it satisfies (2-2). In the same way we see that if the algorithm returns \( \sigma = \sigma_1 \ldots \sigma_n \), then \( \sigma \) is a unifier for \( U \) and \( V \). It remains to prove (2-4), that is,

\[
\theta = \sigma \theta = \sigma_1 \ldots \sigma_n \theta
\]

for any unifier \( \theta \) for \( U \) and \( V \). The proof is by induction on \( n \). Since \( \sigma_1 = \varepsilon \), it holds trivially that

\[
\theta = \sigma_1 \theta
\]

Assume as induction hypothesis

(2-6) \( \theta = \sigma_1 \ldots \sigma_k \theta \)

Let \( \sigma_{k+1} = \{ x = t \} \). It suffices to show

(2-7) \( \sigma_{k+1} \theta = \theta \)

since (2-6) and (2-7) together imply

\[
\theta = \sigma_1 \ldots \sigma_k \theta = \sigma_1 \ldots \sigma_k \sigma_{k+1} \theta
\]

To verify (2-7), it is sufficient to show that \( \sigma_{k+1} \theta \) and \( \theta \) have the same effect on all variables \( v \). Assume first that \( v \) is not \( x \). Then

(2-8) \( v \sigma_{k+1} \theta = v \{ x = t \} \theta = v \theta \)

Next we consider the case where \( v \) is \( x \). Then

(2-9) \( v \sigma_{k+1} \theta = x \{ x = t \} \theta = t \theta \)

Since \( x \) and \( t \) in \( U_\sigma \ldots \sigma_k \) and \( V_\sigma \ldots \sigma_k \) need to be unified, there is a variable \( y \) in \( x \)'s place and a term \( u \) in \( t \)'s place, one of them in \( U \) and the other in \( V \), such that

(2-10) \( x = y \sigma_1 \ldots \sigma_k \)

(2-11) \( t = u \sigma_1 \ldots \sigma_k \)

Since \( \theta \) unifies \( U \) and \( V \), we have
(2-12) \( y \theta = u \theta \)

From (2-6) and (2-10) through (2-12), we infer
\[
v \theta = x \theta = y \sigma_1 \ldots \sigma_k \theta = y \theta = u \theta = u \sigma_1 \ldots \sigma_k \theta = t \theta
\]

which together with (2-9) implies
(2-13) \( v \sigma_{k+1} \theta = v \{x = t\} \theta = t \theta = v \theta \)

in the case where \( v \) is \( x \). The identities (2-8) and (2-13) together imply (2-7).

2.3 Remark. The theorem shows that the unification algorithm 1.4 does what we want it to do: it always finds an MGU in a finite number of steps iff there is one. Therefore this algorithm is conceptually satisfactory. The next result shows that it is not satisfactory with respect to computational complexity.

2.4 Theorem. The algorithm 1.4 is of exponential complexity.

Proof: Consider the unification of the pair of atomic formulas
\[ P(x_1, \ldots, x_n), \quad P(f(x_0,x_0), \ldots, f(x_{n-1},x_{n-1})) \]

using the algorithm. We get \( \sigma_1 = \{x_1 = f(x_0,x_0)\} \) and
\[
P(x_1, \ldots, x_n)\sigma_1 = P(f(x_0,x_0), x_2, \ldots, x_n)
\]
\[
P(f(x_0,x_0), \ldots, f(x_{n-1},x_{n-1}))\sigma_1 =
\]
\[
P(f(x_0,x_0), f(f(x_0,x_0),f(x_0,x_0)), f(x_2,x_2),\ldots,f(x_{n-1},x_{n-1}))
\]

In the next iteration, we have to unify \( x_2 \) and \( f(f(x_0,x_0),f(x_0,x_0)) \) which gives
\[ \sigma_2 = \{x_2 = f(f(x_0,x_0),f(x_0,x_0))\} \]

and similarly for \( \sigma_3, \sigma_4, \ldots \). To accept \( \sigma_1 \), the occur check forces the system to check that \( x_1 \) is not one of the \( 2 = 2^1 \) variable occurrences in \( f(x_0,x_0) \). To accept \( \sigma_2 \), the system must check that \( x_2 \) is distinct from all the \( 4 = 2^2 \) variable occurrences in \( f(f(x_0,x_0),f(x_0,x_0)) \). Generally, \( \sigma_k \) demands \( 2^k \) occur checks. Altogether the unification of the two atomic formulas demands \( 2^{n+1} - 2 \) occur checks so that the algorithm is of exponential complexity.

2.5 Remark. (I) We see that there actually are two sources of complexity in the algorithm. One is, of course, the occur check. But a more fundamental source is the exponential swelling of the lengths of terms in the substitutions
which gives longer and longer expressions for the system to scan when it performs the occur check.

(II) The problem with an exponentially exploding running time for the algorithm is usually solved by simply omitting the occur check from actual implementations, for instance in PROLOG. This step turns out to create less troubles than one might at first expect. Part of the explanation of this is that cases which need to be blocked by the occur check are rather rare. Another part of the explanation is given by the next theorem.

2.6 Theorem. Every recursive function \( f \) is computed correctly by an implementation of PROLOG where the occur check is omitted from the unification algorithm 1.4.

Proof:
The program sentences in the PROLOG program consists of the Horn clauses in the proof of Theorem 1.7 needed to define \( f \). The goal must have the form

\[ \leftarrow f(a) = z \]

Going through the Horn clauses for (RF1)-(RF4), we see inductively that every substitution in an execution of the program must have the form

\[ \{x_1 = t_1, \ldots, x_n = t_n\} \]

where \( t_1, \ldots, t_n \) are variable-free. The occur check cannot block any such substitution. The PROLOG program arrives at exactly the same result for \( f(a) \) without the occur check as when the latter is present.

2.7 Remark. Thus PROLOG without the occur check does all computations of the recursive functions correctly. By the Church-Turing thesis, it also executes all algorithms correctly. Unfortunately this is of limited help. The reason is that there are other programming languages which are considerably better than PROLOG for computations and other algorithms. PROLOG has its strength in automated theorem proving and in processing certain types of databases like expert systems. In such contexts, the absence of the occur check may sometimes be disastrous. It should therefore be helpful to find a sound version of the unification algorithm which is of only polynomial complexity. This problem was solved by Martelli and Montanari (1982) in a remarkable paper. Their strategy is to take Algorithm 1.4, in a suitable reformulation, as their starting point. The algorithm is then refined to a point where the occur check can be made without any exponential explosion in complexity. I now give an exposition and logical analysis of Martelli-Montanari’s work. I have tried to structure the exposition as a logician should want it, and I give careful proofs of the lemmas and theorems. In spite of its
advantages and in spite of its quarter of a century of existence, this algorithm is not well-known among computer scientists and applied logicians. The present essay is, apart from being a logical analysis, also an attempt to make it better known.

2.8 Remark. (I) The problem of unifying the terms $t$ and $u$ can be written as an equation $t = u$. A solution to the equation, a unifier, is any substitution $\sigma$ which makes the two terms identical, that is, $t\sigma$ and $u\sigma$ are the same term.

(II) We also consider sets $S = \{t_i = u_i | i = 1, \ldots, n\}$ of equations. A unifier of $S$ is any substitution $\sigma$ such that $t_i\sigma$ and $u_i\sigma$ are equal for each $i$.

(III) Two sets $S_1$ and $S_2$ of equations are equivalent if they have the same set of unifiers. We consider some operations on sets of equations which leave the associated set of unifiers unchanged; in other words, the transformed set is equivalent with the original set.

2.9 Definition. (Term Reduction). Let $S$ be a set of equations containing an equation of the form

\[(2-14) \quad f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)\]

Obtain $S'$ from $S$ by replacing equation (2-14) with the following $n$ equations:

\[(2-15) \quad t_1 = u_1, \ldots, t_n = u_n\]

Then $S'$ is obtained from $S$ by term reduction. (* If $n = 0$ and $f$ is a constant, then equation (2-14) is erased and no new equation added.*)

2.10 Lemma. Let $S$ be a set of equations and let $f(t_1, \ldots, t_n) = g(u_1, \ldots, u_n)$ be an equation in $S$. If $f$ and $g$ are different function symbols, then $S$ has no unifier. If $f$ and $g$ are the same function symbol and $S'$ is obtained from $S$ by term reduction applied to the given equation, then $S$ and $S'$ are equivalent.

Proof: If $f$ and $g$ are distinct, no unification is possible of the two terms in the equation. If they are identical, any unifier of (2-14) must unify the equations in (2-15), and vice versa.

2.11 Definition (Variable Elimination). Let $S$ be a set of equations containing $x = t$ where $x$ is a variable and $t$ is an arbitrary term. Let $S'$ be the set obtained from $S$ by applying the substitution $\sigma = \{x = t\}$ to all other equations in $S$ while keeping $x = t$ in the set. Then $S'$ is obtained from $S$ by variable elimination.
2.12 Lemma. Let \( S' \) be obtained from \( S \) by applying variable elimination to \( x = t \) in \( S \). Then \( S \) and \( S' \) are equivalent. In particular, if \( x \) occurs in \( t \) and \( t \) is distinct from \( x \), then neither \( S \) nor \( S' \) have unifiers.

\textbf{Proof:}

The last statement follows by the occur check since \( x = t \) belongs to both \( S \) and \( S' \). Let \( S = \{x = t, t_1 = u_1, ..., t_n = u_n\} \) and let \( \sigma \) be the substitution \( \{x = t\} \). Then \( S' = \{x = t, t_1\sigma = u_1\sigma, ..., t_n\sigma = u_n\sigma\} \). First assume that \( \theta = \{y_1 = s_1, ..., y_m = s_m\} \) unifies \( S \). Then

\begin{equation}
(2-16) \quad x\theta = t\theta, \quad t_i\theta = u_i\theta, \ldots, \quad t_n\theta = u_n\theta
\end{equation}

Since \( x\theta = t\theta \), there is an \( i \) such that \( y_i = s_i \) is \( x = t\). By the rule 1.3 for compositions,

\[
\sigma\theta = \{x = t, y_1 = s_1, ..., y_m = s_m\}
\]

\[
= \{y_1 = s_1, ..., y_m = s_m\}
\]

\[
= \theta
\]

so that by (2-16) and the expression for \( S' \), \( \theta \) unifies \( S' \). In the same way, we see that if \( \theta \) unifies \( S' \), then again \( \sigma\theta = \theta \) and \( \theta \) unifies \( S \).

2.13 Definition (Solved form). Let \( S \) be a set of \( k \) equations. \( S \) is in \textit{solved form} if

1. the equations in \( S \) are of the form \( x_j = t_j, j = 1, ..., k \);
2. every variable which is the left member of some equation in \( S \) occurs only there.

2.14 Lemma. If \( S \) is in solved form, then \( S \) is its own unifier, that is, \( \sigma = \{x_1 = t_1, ..., x_k = t_k\} \) unifies \( S \). Moreover, \( \sigma \) is an MGU for \( S \) and for any set equivalent with \( S \).

\textbf{Proof:}

To see that \( \sigma \) is an MGU for \( S \), let \( \theta \) be a unifier for \( S \). Then \( x_i\theta = t_i\theta \) so that

\[
\theta = \{x_1 = t_1\theta, ..., x_k = t_k\theta\} \cup \gamma = \sigma\theta
\]

where \( \gamma \) is a substitution which does not affect \( x_1, ..., x_k \).

2.15 Algorithm. Given a set \( S \) of equations, repeatedly perform any of the following steps.

1. If no transformation applies, \textit{stop} and \textit{return} as MGU the transformed set: \( \sigma = S \).
(2) Select any equation in S of the form \( t = x \), where \( x \) is a variable and \( t \) is not, and replace it by \( x = t \).

(3) Select any equation of the form \( x = x \), where \( x \) is a variable, and remove it from S.

(4) Select any equation in S of the form \( t = u \), where neither \( t \) nor \( u \) are variables. If the root function symbols are different, stop and return False. Otherwise apply term reduction.

(5) Select any equation of the form \( x = t \), where \( x \) is a variable which also occurs in some other equation and \( t \) is not \( x \). If \( x \) occurs in \( t \), then stop and return False. Otherwise apply variable elimination.

2.16 Theorem. Let a set S of equations be given as input to Algorithm 2.15.

(I) The algorithm always terminates.

(II) If the algorithm returns False, then the set S has no unifier. If the algorithm terminates with success and returns a substitution set \( \sigma \), then \( \sigma \) is in solved form, \( \sigma \) is equivalent with the original set S, and \( \sigma \) is an MGU for S.

Proof:

(I): With every set S of equations, we associate an index \( I(S) = (p,q,r) \) where \( p \) is the number of variables in S which occur more than once as the LHS of an equation, \( q \) is the number of occurrences of function symbols in S, and \( r \) is the sum of the number of equations in S of type \( x = x \) and the number of equations of type \( t = x \), where \( x \) is a variable and \( t \) is not. We define an ordering on \( \mathbb{N}^3 \):

\[
(p,q,r) < (p',q',r') \iff p < p' \text{ or } (p = p' \text{ and } q < q') \text{ or } (p = p' \text{ and } q = q' \text{ and } r < r')
\]

Then \( \mathbb{N}^3, < \) is clearly a well-ordered set and therefore has no infinitely descending chain. It therefore suffices to show that if \( S' \) is the result of applying one of the transformations (2)-(5) in Algorithm 2.15 to S, then \( I(S') < I(S) \). This is, indeed, the case since any of transformations (2) and (3) decreases \( r \) without increasing \( p \) and \( q \). Step (4) decreases \( q \) (by 2) and does not increase \( p \). Step (5), finally, decreases \( p \).

(II): If the algorithm terminates with output False, the original set S has no unifier by lemmas 2.12 and 2.14. If the algorithm terminates with success and the set of equations \( \sigma \) as output, then \( \sigma \) (= the final form of the set S) is equivalent with the original form of S. This follows since steps (2) and (3) trivially do not change the set of unifiers. For steps (4) and (5), the conclusion is given by lemmas 2.12 and 2.14. Lemma 2.14 also implies that \( \sigma \) is an MGU for the original S. Finally, \( \sigma \) is in solved form by Definition 2.13. For if none of transformations (2)-(4) applies, then all equations are in the form
x = t where t is not x. If step (5) does not apply, every variable in the LHS of some equation occurs only there in the final form of S.

2.17 Remark. (I) Algorithm 2.15 is an indeterministic algorithm. It is trivial to transform it into a deterministic algorithm by adding data structures and unambiguous instructions concerning the order in which transformations should be made and the order in which equations should be selected. Even the following algorithms in the present section will be given in indeterministic form. This allows us to concentrate on the essentials in the algorithms. Transforming an algorithm into deterministic form does not essentially raise its degree of complexity. Thus a deterministic version of an indeterministic linear algorithm will also be linear.

(II) The unification algorithms 1.4 and 2.15 are variants of each other. Thus Algorithm 2.15 is also of exponential complexity. It is only a first step towards a faster unification algorithm. We now take the next step. The idea is to introduce the concept of a multiequation and adapt the Algorithm 2.15 to multiequations.

2.18 Definition (Multiequations). (I) A multiset is a collection of elements among which no ordering exists but identical elements may occur in more than one place. We use the notation \((t_1, \ldots, t_n)\) for a finite multiset. (* Thus \((a, b, c) = (b, a, c) \neq (b, a, c, a) = (a, a, b, c).*\)

(II) A multiequation is an expression of the form \((S = M)\) where \(S \neq \emptyset\) is a set of variables and \(M\) is a multiset of terms none of which is a variable.

(III) A solution (unifier) of a multiequation \((S = M)\) is a substitution which unifies all terms occurring in \(S\) and \(M\).

2.19 Example. An example of a multiequation is
\[
\{x, y, z\} = (t, u)
\]
if \(t\) and \(u\) are not variables. A multiequation is a way of grouping several equations together. For example the set of equations
\[
S = \{x = y, z = x, t = x, y = u\}
\]
can be transformed into the above multiequation because the multiequation and the set \(S\) of equations have the same unifiers. I now give a precise definition of this relation.

2.20 Notation. Let \(E\) be a set of equations. Let \(T_E\) be the set of terms occurring as LHS or RHS in some equation in \(E\). Define a sequence \(\{E_n\}\) by
\begin{align*}
E_0 &= E \cup \{ t = t \mid t \in T_E \} \\
E_{n+1} &= E_n \cup \{ u = t \mid (t = u) \in E_n \} \\
&\quad \cup \{ t_1 = t_3 \mid (t_1 = t_2), (t_2 = t_3) \in E_n \text{ for some } t_2 \in T_E \}
E^* &= \bigcup_n E_n
\end{align*}

If \( E \) is finite, then \( E_{k+1} = E_k \) for some \( k \) so that also \( E^* \) is finite. \( E^* \) is essentially the reflexive, symmetric, and transitive closure of \( E \).

2.21 **Definition.** A set of equations \( E \) corresponds to a multiequation \((S = M)\) iff \( S \cup M = T_E \) and for every \( t, u \in S \cup M \), we have \((t = u) \in E^*\).

2.22 **Lemma.** (I) Several sets of equations may correspond to a given multiequation \((S = M)\). Indeed, let \( E \) correspond to \((S = M)\), as in the definition, and let \( F \) be a set of equations. Then \( F \) corresponds to \((S = M)\) iff \( F^* = E^* \)

(II) If \( E \) corresponds to \((S = M)\), then \( E \) is equivalent with \((S = M)\), that is, \( E \) and \((S = M)\) have the same set of unifiers.

(III) Let \( F = \{ x = t \mid x \in S \land t \in M \} \). Then the set \( E \) of equations corresponds to \((S = M)\) iff \( E^* = F^* \). In particular, \( F \) corresponds to \((S = M)\).

**Proof:**

(I) \( \Rightarrow \): Then \( T_F = S \cup M = T_E \), and

\[ E^* = \{ t = u \mid t, u \in T_E \} = \{ t = u \mid t, u \in T_F \} = F^* \]

(I) \( \Leftarrow \): Then \( T_F = T_F^* = T_E^* = T_E = S \cup M \), and for all \( t, u \in S \cup M \), \((t = u) \in F^* = E^* \). Therefore \( F \) corresponds to \((S = M)\).

(II): From the definition, it is clear that \( \sigma \) unifies the terms in \( S \cup M \) iff \( \sigma \) unifies the terms in \( T_E \).

(III): From the definition, we see that \( F \) corresponds to \((S = M)\). By Part (I) of the present lemma, the equivalence in (III) follows.

2.23 **Definition.** Let \( Z = \{ (S_1 = M_1), \ldots, (S_n = M_n) \} \) be a set of multiequations and let \( E \) be a set of equations. Define

\[ F = \{ x = t \mid \exists i (x \in S_i \land t \in M_i) \} \]

Then \( E \) corresponds to \( Z \) iff \( E^* = F^* \).

2.24 **Lemma.** Let \( E \) correspond to \( Z \) as in the definition. Then \( E \) and \( Z \) have identical sets of unifiers, that is, \( E \) and \( Z \) are equivalent.
Proof:
If $\sigma$ unifies $E$, then it unifies $F$ by Lemma 2.23 where $F$ is as in Definition 2.22. From the definition of $F$ we see that $\sigma$ unifies each $(S_i = M_i)$ so that $\sigma$ unifies $Z$. Conversely, if $\sigma$ unifies $Z$, it unifies all $(S_i = M_i)$ and hence $F$. Then it unifies the equivalent $E$.

2.25 Remark. Our next goal is to reformulate Algorithm 2.15 so that it is adapted to sets of multiequations as input rather than sets of equations. For the new algorithm, we need two transformations of sets of multiequations: multiequation reduction and compactification. To define them, we need two new concepts: the common part of a multiset $M$ and the frontier of $M$.

2.26 Definition (Common part). Let $M = (t_1, ..., t_n)$ be a multiset of terms. The common part $C(M)$ of $M$ is a term $t$ such that

1. every variable occurring in $t$ also occurs in the same place in some term $t_i$ in $M$;
2. for each $i = 1, ..., n$ there is a substitution $\sigma_i$ which gives $t_i = t_{\sigma_i}$.

2.27 Remark. (I) All common parts of $M$ are variants of each other, which justifies the expression "the common part of $M$".

(II) $C(M)$ may not exist. In that case $M$ cannot be unified.

2.28 Definition. Let $M = (t_1, ..., t_n)$ be a multiset of terms. The frontier $F(M)$ of $M$ is a set of multiequations obtained as follows. Let $x_1, ..., x_m$ be all the variables in $C(M)$. For $i = 1, ..., m$, form the multiequation

\[(2\cdot17) \quad \{x_i\} = (x_i\sigma_1, ..., x_i\sigma_n)\]

where the $\sigma_j$ are the substitutions in Definition 2.26 which give $t_j = C(M)\sigma_j$. For $j = 1, ..., n$, if $x_i\sigma_j$ is a variable, move it to the LHS of (2\cdot17). Let $F(M)$ contain the resulting set of multiequations.

2.29 Remark. The common part $C(M)$ and the frontier $F(M)$ can be found by an algorithm. We define a recursive procedure $CF$ which for input $M$, a multiset of terms, returns either False, in which case $M$ has neither common part nor frontier, or else returns $C(M)$ and $F(M)$.

2.30 Notation. We use the following notation in the formulation of the procedure $CF$.
If $t$ is not a variable, then $head(t)$ denotes the root function symbol of $t$.
If $f$ is an $n$-place function symbol and $1 \leq i \leq n$, then $P_i$ is the $i^{th}$ projection defined by
makemulteq is a function defined by

makemulteq(M) = (S' = M')

where M is a multiset of terms of which at least one is a variable and (S' = M') is a multiequation such that S' is the set of all variables in M and M' is the multiset of all non-variable terms in M.

An = the set of all n-place function symbols.

2.31 Procedure (CF(M)). Let M = (t1, ..., tn) be an arbitrary multiset of terms.

if M contains a variable x
then return C(M) := x, F(M) := {makemulteq(M)}
else if \exists n (\exists f \in A_n) (\forall t \in M) head(t) = f
then if n = 0
then return C(M) := f, F(M) := \emptyset
else if \forall i (1 \leq i \leq n) \exists f_i \in A_n \forall t \in M head(t) = f_i
then return C(M) := f(C(M_1), ..., C(M_n)), F(M) := \cup_i F(M_i)
else return False
else return False.

2.32 Remark. It is easy to check that the procedure CF does, indeed, find the common part and the frontier as in definitions 2.27 and 2.29. I now define multiequation reduction and compactification.

2.33 Definition (Multiequation reduction). Let Z be a set of multiequations and (S = M) an element of Z such that M \neq \emptyset has a common part C(M) and a frontier F(M). Obtain Z' from Z by removing (S = M) and replacing it by Z' = (Z - {S = M}) \cup {S = (C(M))} \cup F(M)

Then Z is transformed into Z' by multiequation reduction on (S = M).

2.34 Lemma. Let Z be a set of multiequations containing (S = M) where M \neq \emptyset. Let Z' be obtained from Z by multiequation reduction on (S = M).

(I) If M has no common part, then Z has no unifier.
(II) If some variable in S belongs to the LHS of some multiequation in F(M), then Z has no unifier.
(III) Z and Z' are equivalent.
Proof:

(I): From the procedure CF, we see that if M has no common part only if there is a clash of function symbols in M, in which case \( S = M \), and hence Z, cannot be unified.

(II): If the variable \( x \in S \) occurs on the LHS of a multiequation in \( F(M) \), then we see from definitions 2.26 and 2.28 that \( x \) occurs in a non-variable term in M. It follows that \( S = M \) has no unifier and then neither has \( Z \).

(III): Assume that \( \sigma \) is a unifier for \( Z \). Then, a fortiori, \( \sigma \) is a unifier for \( (Z - \{S = M\}) \). Let \( x \) occur in \( C(M) \). Then \( x \) occurs in some \( t_k \) in the same place so that \( x_\sigma t_k = x \) and hence \( x_\sigma t_\sigma = x_\sigma \). Now consider an arbitrary \( \sigma_\tau \). Since \( C(M)\sigma_\tau = C(M)\sigma_\tau_\sigma \), we have, for the subterm \( x \), that \( x_\sigma t_\sigma = x_\sigma \). For any \( y \) occurring in \( M \) but not in \( C(M) \), we have \( y_\sigma = y \) so that \( y_\sigma_\tau = y_\tau \). Therefore for all variables \( x \) in \( M \) and for all \( i = 1, ..., n \), \( x_\sigma t_\sigma = x_\sigma \).

Then \( \sigma \) unifies \( S = C(M) \) and also all multiequations in \( F(M) \) since they are derived from multiequations of the form

\[
\{x_i\} = (x_1\sigma_1, \ldots, x_n\sigma_n)
\]

2.35 Definition (Compactness). Let \( Z \) be a set of multiequations. \( Z \) is compact if for all \( (S = M) \) and \( (S' = M') \) in \( Z \), if \( S \neq S' \), then \( S \cap S' = \emptyset \).

2.36 Definition (Compactification). Let \( Z \) be a set of multiequations. Let \( Z' = \{[S = M] | (S = M) \in Z\} \) where \( [S = M] \) is a subset of \( Z \) defined by \( (S' = M') \in [S = M] \) iff there is a sequence \( (S_1 = M_1), \ldots, (S_n = M_n) \in Z \) such that \( S_1 = S, M_1 = M, S_n = S', M_n = M', \) and \( S_i \cap S_{i+1} \neq \emptyset \) for \( 1 \leq i < n \). Then \( Z \) is a partition of \( Z \). Obtain \( Z' \) from \( Z \) by putting a multiequation \( (S' = M') \) into \( Z' \) for each cell \( \{S_1 = M_1, \ldots, S_n = M_n\} \) in \( Z \) where \( S' = \cup S_i \) is the union of the \( S_i \) and \( M' \) is the union of the \( M_i \). We say that \( Z' \) is derived from \( Z \) by compactification.

2.37 Lemma. Let \( Z' \) be derived from \( Z \) by the compactification operation 2.36. Then

(I) \( Z' \) is compact.

(II) \( Z \) and \( Z' \) are equivalent.
**Proof:**

**(I):** Let the multiequations \( S_1 = M_1 \) and \( S_2 = M_2 \) be in \( Z' \) with \( S_1 \neq S_2 \). Let \( S_1 = M_1 \) be obtained from the cell \( \{S_{11} = M_{11}, \ldots, S_{1p} = M_{1p}\} \) by taking unions, and \( S_2 = M_2 \), similarly, obtained from \( \{S_{21} = M_{21}, \ldots, S_{2q} = M_{2q}\} \). Then \( S_{1i} \cap S_{2j} = \emptyset \) for all \( i, j \), and hence, by distributivity,

\[
S_1 \cap S_2 = (\bigcup_i S_{1i}) \cap (\bigcup_j S_{2j}) = \emptyset
\]

**(II):** Define \( F \) for \( Z \) and \( F' \) for \( Z' \) as in Definition 2.23. Then clearly \( F^* = F'^* \), and \( F \) corresponds to \( Z \) and \( F' \) to \( Z' \). By Lemma 2.24, \( Z \) and \( Z' \) are equivalent.

2.38 **Remark.** Algorithm 2.15 takes sets \( E \) of equations as input. It then transforms \( E \) step-by-step into solved form. The solved form of \( E \) is a unifier for the original set \( E \). We now want to formulate an analogous algorithm which takes sets of multiequations as input. For this purpose, it is useful to introduce the concept of a system of multiequations \( R = (T, U) \). Initially, when \( R \) is an input to the algorithm, \( T \) is empty while \( U \) is the set of multiequations to be unified. Each multiequation in \( U \) is then transferred to \( T \) after having been brought into solved form.

2.39 **Definition (Systems of multiequations).** (I) A *system of multiequations* is an ordered pair \( R = (T, U) \) where \( T \) is a sequence of multiequations and \( U \) is a set of multiequations which satisfy the following conditions:

1. The sets of variables which constitute the LHSs in \( T \) and \( U \) are pairwise disjoint and contain all variables occurring in \( T \) and \( U \);
2. the RHS of any multiequation in \( T \) contains at most one term;
3. the variables belonging to the LHS of some multiequation \( (S = M) \) in \( T \) occurs only on the RHS of multiequations preceding \( (S = M) \) in \( T \).

(* Recall from Definition 2.19 that no multiequation \( (S = M) \) has a variable as element in \( M \).*

II) \( T \) is the *solved part* of \( R \), \( U \) is the *unsolved part*. If \( U = \emptyset \), then \( R \) is in *solved form*.

2.40 **Remark.** From the example in the proof of Theorem 2.4, we see that there are two sources of the exponential complexity of the unification algorithm. One is the occur check; but a more basic source is the exponential swelling of the size of the MGUs \( \sigma_1, \sigma_2, \ldots \). The use of systems of multiequations starting with \( R = (T, U) \), where \( T \) is empty and \( U \) contains terms to be unified, and ending with \( R' = (T', U') \), where \( U' \) is empty and \( R' \) is in solved form, gives a way to avoid this phenomenon.
2.41 **Algorithm.** Let a system \( R = (T, U) \) with \( T = (\ ) \) be given as input.

(1) **repeat**
   
   (1.1) Select a multiequation \((S = M)\) from \( U \) with \( M \neq \emptyset \).
   
   (1.2) Compute \( C(M) \) and \( F(M) \). If \( M \) has no common part, **stop** and **return** \( \text{False} \).
   
   (1.3) If a LHS of a multiequation in \( F(M) \) contains a variable also element in \( S \), **stop** and **return** \( \text{False} \).
   
   (1.4) Transform \( U \) by applying multiequation reduction on \((S = M)\) and then compactification.
   
   (1.5) Let \( S = \{x_1, \ldots, x_n\} \). Apply the substitution \( \sigma = \{x_1 = C(M), \ldots, x_n = C(M)\} \) to the RHS of each multiequation in \( U \).
   
   (1.6) Transfer the multiequation \( S = C(M) \) from \( U \) to the end of \( T \).
   
   until \( U = \emptyset \) or all multiequations in \( U \) have empty RHS.

(2) Transfer all the multiequations of \( U \) with \( \text{RHS} = \emptyset \) to the end of \( T \), **stop** and **return** \( R = (T, U) \).

2.42 **Remark.** (I) Algorithm 2.41 is still of exponential complexity. Step (1.5) is a source of complexity since it may give rise to many copies of large terms.

(II) If we want to unify the terms \( t_1, \ldots, t_m \), we must, as input to the algorithm, construct an initial system \( R = (T, U) \) with

\[
\begin{align*}
T: & (\ ) \\
U: & \{x, y_1, \ldots, y_k\} = \{u_1, \ldots, u_p\} \\
\{x_1\} = \emptyset, \ldots, \{x_n\} = \emptyset
\end{align*}
\]

where \( x \) is a new variable not occurring in any of \( t_1, \ldots, t_m \); \( y_1, \ldots, y_k \) are those terms among \( t_1, \ldots, t_m \) which are variables and \( u_1, \ldots, u_p \) are those which are not; \( x_1, \ldots, x_n \) are all the variables occurring in \( u_1, \ldots, u_p \) but not in \( \{x, y_1, \ldots, y_k\} \). All of this can be done mechanically and made part of the algorithm.

2.43 **Theorem.** Let the multiequation system \( R = (T, U) \) with \( T = \emptyset \) be given as input to Algorithm 2.41.

(I) The algorithm always terminates.

(II) If it returns \( \text{False} \), then the system \( R \) has no unifier.

(III) If it stops with success and returns \( R' = (T', U') \), then the output \( R' \) is a system equivalent with the input \( R \), the input system is unifiable, \( U' = \emptyset \) and \( R' \) is in solved form.
Proof:
I prove the claims in the order (III), (I), (II).

(III): I show that if \( R = (T, U) \) is a system and \( R \) during one iteration is transformed into \( R' = (T', U') \), then \( R' = (T', U') \) is a system and \( R' \) is equivalent with \( R \). Since no variable is erased from the LHS of any multiequation in \( U \), the LHSs of the multiequations in \( T' \) and \( U' \) contain all the variables in \( T \) and \( U \). The presence of Step (1.3) together with the fact that compactification is done by taking unions of cells in a partition of \( U \), as in Definition 2.36, guarantees that the LHSs of the multiequations in \( T' \) and \( U' \) are pairwise disjoint. Thus the first condition in Definition 2.39 is satisfied. Since, at each iteration, the multiequation transferred from \( U \) to \( T \) has the form \( S = (C(M)) \), even the second condition in the definition of a system is satisfied. If \( x \) occurs in \( S \), then \( x \) cannot occur in the LHS of any other multiequation in \( T \) because of the disjointness. Suppose \( x \) occurs in \( C(M) \). Then, as in the proof of Lemma 2.34, \( x \sigma_k = x \) for some \( k \) so that \( x \) occurs on the LHS of some multiequation in \( F(M) \), which is excluded by step (1.3). Thus no variable \( x \) in \( S \) occurs in \( C(M) \). Moreover, by Step (1.5), \( x \) is eliminated from \( U \) and cannot occur in any later multiequation in \( T \) either. It can only occur in the RHS of multiequations preceding \( S = C(M) \) in \( T \). Thus also Condition (3) in Definition 2.39 is satisfied. To show that \( T' \) and \( U' \) contain no multiequation-like expression with a variable as member of its RHS, we note that the only ways of introducing new multiequations into \( U \) are by the operation of taking the frontier of \( M \) and the operation of compactification. By construction, no element in \( F(M) \) has a variable member in its RHS. Compactification produces new multiequations by taking unions of LHSs and unions of RHSs of already existing multiequations and therefore cannot result in a variable as a member of a RHS either. We have now shown that \( R' \) is, indeed, a system of multiequations. Finally we verify that \( R \) and \( R' \) are equivalent. We note that Step (1.4) preserves equivalence by Lemmas 2.34 and 2.37. Since \( S = (C(M)) \) is kept in the system, Step (1.5) cannot, by Lemma 2.12 on variable elimination, destroy the equivalence. Step (1.6) preserves equivalence because, having passed test (1.3), \( S = (C(M)) \) is not modified by compactification. Step (2) trivially has no effect on equivalence and keeps \( R \) as a system of multiequations. The rest of Claim (III) now follows immediately.

(I): The algorithm always terminates because if it does not terminate with \textbf{False} as output, then, after each iteration of step (1), the number of variables in \( U \) decreases with at least one.

(II): If, in some iteration, the algorithm returns \textbf{False} because \( M \) has no common part (at Step (1.2)), then there is a clash of function symbols. As a consequence, \( (S = M) \) cannot be unified and hence neither can the input \( R = (T, U) \) by equivalence. Now suppose that the algorithm returns \textbf{False} at Step
(1.3) because a LHS of some multiequation in F(M) contains a variable z also in S. Then the multiequation is derived from identity (2-17) in Definition 2.28 for some i,
\[ \{x_i\} = (x_i \sigma_1, \ldots, x_i \sigma_n) \]
so that either z is \( x_i \) or \( z = x_i \sigma_j \) for some j. In the first case, z occurs in some \( t_k \) in M, and in the second case in \( t_j \). Since neither \( t_j \) nor \( t_k \) are variables, \( (S = M) \) cannot be unified. By equivalence, neither can the input system \( R = (T, U) \).

2.44 **Remark.** The main source of complexity in Algorithm 2.41 is the substitution (1.5) since it may result in many copies of large terms. We now show that if \( R = (T, U) \) is unifiable and \( U \neq \emptyset \), then there always is a multiequation \( (S = M) \) in U such that if \( (S = M) \) is selected, then we do not need Step (1.5) because the variables in S do not occur elsewhere in U.

2.45 **Definition.** Let a system \( R = (T, U) \) with \( U = \{S_i = M_i \mid 1 \leq i \leq n\} \) be given. Define \( S_i < S_j \) iff \( S_i \) contains a variable which occurs in a term in \( M_j \). Let \( <^* \) denote the transitive closure of \(<\).

2.46 **Lemma.** If the system \( R = (T, U) \) is unifiable, then \(<^* \) is a (strict) partial ordering.

**Proof:**
We must show that \(<^* \) is asymmetrical and transitive. The relation is transitive by definition. Suppose \(<^* \) is not asymmetrical. Then there are i, j such that \( S_i <^* S_j \) and \( S_j <^* S_i \). By transitivity, \( S_i <^* S_i \). We then have a cycle \( S_i < S_j < \ldots < S_k < S_i \). Let \( \sigma \) be a unifier for \( R \). Then \( \sigma \) unifies \( \{S_i = M_i, S_j = M_j, \ldots, S_k = M_k\} \). Since the RHSs contain no variables, it follows that the length of the unified terms in \( M_i \) is shorter than the length of the unified terms in \( M_j \) etc., which are shorter than the length of the unified terms in \( M_k \), which are shorter than the length of the unified terms in \( M_i \), which is impossible. Therefore \(<^* \) must be asymmetrical.

2.47 **Lemma.** If the system \( R = (T, U) \) is unifiable and \( U \neq \emptyset \), then there is a multiequation \( (S = M) \in U \) such that the variables in S do not occur elsewhere in U.

**Proof:**
Since \( \Sigma = \{S_i \mid (S_i = M_i) \in U\} \) is a finite partially ordered set, there is an \( (S = M) \) in U such that for none of the \( S_i \) do we have \( S <^* S_i \), that is, \( S \) is a maximal element of \( \Sigma \). Suppose \( S \) contains a variable \( x \) which also occurs in some \( M_j \). Then \( S < S_j \) and hence \( S <^* S_j \), contrary to the choice of \( S \). Since
the $S_i$ are pairwise disjoint, no variable in $S$ occurs anywhere else in another LHS in $U$ either.

2.48 Algorithm (Martelli-Montanari Unification Algorithm). Let a system $R = (T, U)$ with $T = (\ )$ be given as input.

(1) \begin{align*}
\text{repeat} \\
\hspace{1cm} (1.1) & \text{Select a multiequation } (S = M) \text{ in } U \text{ such that no variable in } S \text{ occurs elsewhere in } U. \text{ If no such multiequation exists,} \\
\hspace{1cm} & \text{stop and return } \text{False}. \\
\hspace{1cm} (1.2) & \text{if } M = \emptyset \\
\hspace{1cm} & \text{then transfer } (S = M) \text{ to the end of } T. \\
\hspace{1cm} & \text{else begin} \\
\hspace{1cm} & \hspace{1cm} (1.2.1) \text{Compute } C(M) \text{ and } F(M). \text{ If } M \text{ has no common part,} \\
\hspace{1cm} & \hspace{1cm} \text{stop and return } \text{False}. \\
\hspace{1cm} & \hspace{1cm} (1.2.2) \text{Transform } U \text{ by multiequation reduction on } (S = M) \text{ and} \\
\hspace{1cm} & \hspace{1cm} \text{compactification.} \\
\hspace{1cm} & \hspace{1cm} (1.2.3) \text{Transfer } (S = C(M)) \text{ from } U \text{ to the end of } T. \\
\hspace{1cm} & \text{end} \\
\hspace{1cm} & \text{until } U = \emptyset \\
(2) & \text{Stop with success and return } R = (T, U). 
\end{align*}

2.49 Remark. The differences between the algorithms 2.41 and 2.48 are the following.

(1) The Martelli-Montanari algorithm (the MM-algorithm) is, in one respect, more restrictive at Step (1.1) in selecting the multiequation $(S = M)$. The existence of a multiequation satisfying the restriction is guaranteed by Lemma 2.47 whenever $R = (T, U)$ is unifiable.

(2) Step (1.3) in Algorithm 2.41 is omitted. The corresponding check is made already in Step (1.1) in Algorithm 2.48.

(3) Step (1.5) in Algorithm 2.41 has also been omitted. Since the variables in $S$ occur nowhere else in $U$, no substitutions as in (1.5) can be made in the MM-algorithm. Therefore nothing is lost by omitting this step.

(4) The MM-algorithm allows the selection of a multiequation $(S = M)$ with $M = \emptyset$ at Step (1.1). It is no longer needed to wait until Step (2) before such a multiequation is transferred to $T$. This was required in Algorithm 2.41 to guarantee that no troubles could arise when we substitute backward in $T$ to get the MGU. In the MM-algorithm, the $\prec$-relation guarantees that $(S = \emptyset)$ cannot be selected before such troubles can no longer arise.
2.50 **Theorem.** Let the system $R = (T, U)$ with $T = \emptyset$ be given as input to the MM-algorithm 2.48.

(I) The algorithm always terminates.

(II) If it returns **False**, then $R$ has no unifier.

(III) If it stops with success and returns $R = (T, U)$, then the output $R$ is equivalent with the input, the input is unifiable, the output component $U$ is empty, and $R$ is in solved form.

(IV) The MM-algorithm is of second order polynomial complexity.

**Proof:**

Most of the theorem follows from Theorem 2.43 and Remark 2.49. Of course, if some multiequation in $F(M)$ contains a variable also in $S$, then some variable in $S$ also occurs somewhere else in $U$. Therefore the occurrence check at Step (1.3) in Algorithm 2.41 is, in the MM-algorithm, performed already when $(S = M)$ is selected. This justifies the omission of Step (1.3).

(IV): Step (1.2.3) certainly does not change the size of $R$. For Step (1.2.2) and multiequation reduction, we see from equation (2-17) in Definition 2.28 that the right-hand sides of $\{S = C(M)\} \cup F(M)$ together are of at most the same size as $M$. This remains true after compactification. The joint size of the left-hand sides in $R$ is constant while the joint size of the left-hand sides in $U$ decreases during each iteration. By the proof of Theorem 2.43(I), the number of variables in $U$ decreases with at least one for each repetition of Step (1). If the joint size of the original $U$ given as input to the algorithm is $x$, then the number of iterations of Step (1) can be at most $x$ so the total number of symbols to be scanned by the algorithm before halt can be at most of the order $x^2$. It follows that the MM-algorithm is of at most second order polynomial complexity.

2.51 **Summary.** Here are the main steps in the formulation of the Martelli-Montanari unification algorithm.

(I) First the unification problem is formulated as the problem of solving a set of equations $\{t_i = u_i | i = 1, ..., n\}$ where the $t_i$ and $u_i$ are terms. Robinson's unification algorithm is adapted to this formulation (Algorithm 2.15).

(II) The concepts of multiequation and system of multiequations are defined. Algorithm 2.41, which is still of exponential complexity, is formulated for inputs $R = (T, U)$ of systems.

(III) Finally we reap the fruits of using systems of multiequations $R = (T, U)$ as input. A partial order can be defined among the left-hand sides of the multiequations in $U$. Via this partial order, Algorithm 2.41 is modified to get a unification algorithm 2.48 of polynomial complexity.
2.52 **Remark.** (I) Algorithm 2.48 can be refined in different ways. It can be implemented. Martelli and Montanari give a formulation of it in PASCAL. All of this belongs to practical programming rather than applied logic so we skip it here.

(II) Martelli and Montanari and others have made experimental tests of the MM-algorithm and found that it does well in comparison with other competing unification algorithms. They have also shown that the algorithm is highly efficient and time saving in an automatic theorem prover based on resolution.

2.53 **Note.** The proof of Theorem 2.2 is a slight simplification of the textbook proofs in Fitting (1996) and Nerode and Shore (1997). The example in the proof of Theorem 2.4 is from Nerode and Shore. The proof of Theorem 2.6 is adapted from Nerode and Shore. In the rest of Section 2, from § 2.8 and on, I follow Martelli and Montanari (1982), with several modifications. In particular, I have tried to give careful proofs of all lemmas and theorems. Definitions 2.20, 2.23, 2.26, 2.28, 2.35 and 2.36 are modified and partly new. Lemmas 2.22, 2.24, and 2.37 and Theorem 2.50 are new or partly new. The proofs of lemmas 2.12 and 2.34 and of Theorem 2.43 are partly new.

**References**


