Matching and Unification in Rewrite Theories

Subrata Mitra
Application Development Technology Institute
IBM Software Solutions Division
555 Bailey Avenue, San Jose, CA 95141
mitra@vnet.ibm.com
Phone [+1] 408-463-4276

Nachum Dershowitz
Department of Computer Science
University of Illinois
Urbana, IL 61801, U.S.A.
nachum@cs.uiuc.edu
Phone [+1] 217-333-4219

January 1996

Abstract

"Semantic unification" is the process of generating a basis set of substitutions (of terms for variables) that makes two given terms equal in a specified theory. Semantic unification is an important component of some theorem provers. "Semantic matching," a simpler variant of unification, where the substitution is made in only one of the terms, has potential usage in programming language interpreters.

Decidable matching is required for pattern application in pattern-directed languages, while decidable unification is useful for theorem proving modulo an equational theory.

In this paper we restrict ourselves to matching and unification problems in theories that can be presented as convergent rewrite systems, that is, finite sets of equations that compute unique output values when applied (from left-to-right) to input values. The new results presented here, together with existing results, provide a much finer characterization of decidable matching and unification than was available before.

1 Introduction

Equation solving is the process of finding a substitution (of terms for variables) that makes two terms equal in a given theory, while *semantic unification* is the process which generates a basis set (for any solution to a given goal, the basis set must contain an element that is equivalent in the underlying theory to one that is at least as general) of such unifying substitutions. A simpler version of this problem, *semantic matching*, restricts the substitution to apply only to one of the terms (called the *pattern*). While semantic unification is used in some theorem provers for performing deductions modulo an equational theory (the theory of associativity and commutativity is a prime example), semantic matching has potential applications in pattern-directed languages. For example, in a functional language, we may define *append* and *reverse* on lists using the following equations:

$$\begin{gathered} append(nil, x) = x,\ append(x \cdot y, z) = x \cdot append(y, z) \\ reverse(nil) = nil,\ reverse(x \cdot y) = append(reverse(y), x \cdot nil) \end{gathered}$$

where *nil* denotes the empty list, and $x \cdot y$ represents a list with *head* x and *tail* y. Given such a set of equations, we could solve a query of the form $x \overset{?}{=} append(nil, nil)$ by evaluating the right-hand side expression $append(nil, nil)$, using the equations as left-to-right rewrite rules, to yield the solution $\{x \mapsto nil\}$. Given these equations, the following definition of *palindromes* comes for free:

$$\begin{aligned} palindrome(append(x, reverse(x))) &= true \\ palindrome(append(x, y \cdot reverse(x))) &= true \end{aligned}$$

In order to use such definitions in a functional language, it is necessary to match patterns of the form $append(x, reverse(x))$ to values like $1 \cdot 2 \cdot 2 \cdot 1 \cdot nil$. To perform such matchings (which is not possible in current functional languages), a decidable matching algorithm is required. Similarly, if we could unifying with respect to *append*, we could solve queries of the form $append(x, x) \overset{?}{=} x$ in logic-programming languages.

We will say that a rewrite system is *convergent* (technically, *ground* convergent) if every ground term has exactly one normal form. For such systems, every reducible substitution is equivalent in the theory to an irreducible one; hence, one can ignore reducible solutions to semantic unification and matching problems. For example, for a goal like $append(1 \cdot x, 2 \cdot nil) \overset{?}{=} 1 \cdot 2 \cdot nil$, in the theory mentioned above, the only solution of interest is, $\{x \mapsto nil\}$ (and not $\{x \mapsto append(nil, nil)\}$, and so forth). It is well-known that any strategy for finding a complete set of unifiers, or of matchings, for two terms, with respect to a given theory, may not terminate, even when the theory is presented as a finite and *convergent* (terminating and confluent) set of rewrite rules; see, for example, [Heilbrunner and Hölldobler, 1987; Bockmayr, 1987]. On the other hand, for some special classes of theories—associativity, for instance—semantic unification is decidable.

In this paper, we are interested in unification and matching in theories that have a convergent presentation. Since the general matching and unification problems with convergent systems are known to be undecidable, we are interested in characterizing restricted convergent systems (using mainly simple syntactic measures) for which, either

- the unification and/or matching problem is decidable, which constitutes the positive cases (i.e., restrictions over and above convergence results in decidability), or
- the unification and/or matching problem continues to be undecidable, which constitutes negative results.

Thus, as such, we are not interested in specific theories such as associativity and commutativity. Our aim is to be able to characterize classes of rewrite systems which has either a decidable unification/matching problem, or for which the problem(s) is (are) known to be undecidable. To prove undecidability, we will use a single counterexample, but will ensure that the corresponding presentation of the theory obeys the required restrictions. The only decidability proof of this paper would use a well-founded ordering to show that a complete procedure for the required problem is also terminating.

We use standard terminology and notations about rewrite systems. Missing definitions can be found in [Dershowitz and Jouannaud, 1990], which is a survey of the field. Given a set $\mathcal{F}$ of function symbols and a (denumerable) set $\mathcal{X}$ of variables, the set of (first-order) *terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the smallest set containing $\mathcal{X}$ such that $f(t_1, \ldots, t_n)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ whenever $f \in \mathcal{F}$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for $i = 1, \ldots, n$. A term t is said to be *linear* in a variable x if x occurs exactly once in t, while a term is *linear* if it is linear with respect to each of its variables, for example, $x + (s(y) * z)$. The *depth* of a term is the length of the longest path in its tree representation.

Reasoning with equations requires replacement of subterms by other terms. A *substitution* is a special kind of replacement operation, uniquely defined by a mapping from variables to terms which is equal to identity almost everywhere, and written out as $\{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$.

A *rewrite rule* is an ordered equation between terms, written as $l \to r$, for terms l and r. A rule is (*left-*) *right-linear* is its (left-) right-hand side is linear, it is *linear* if it is both left- and right-linear, and is *non-erasing* if every variable in l also appears in r. A *rewrite system* is a finite set of rewrite rules. We use $\to$ to denote a single step of derivation (application of a rewrite rule), and $\to^*$ as its reflexive-transitive closure. A term s is said to be *irreducible* or in *normal form* if there is no term t such that $s \to t$. We write $s \to^! t$ if $s \to^* t$ and t is in normal form, and we say that t is the normal form of s. A rewrite relation $(\to)$ is *terminating* if there exists no infinite chain of rewrites of the form $t_1 \to t_2 \to \ldots \to t_k \ldots$. A rewrite relation is (*ground*) *confluent* if, whenever two (ground) terms s and t are derivable from a term u, then a term v is derivable from both s and t, that is, if $u \to^* s$ and $u \to^* t$ then there must be a term v such that $s \to^* v$ and $t \to^* v$. A rewrite system which is both terminating and (ground) confluent is said to be (*ground*) *convergent.*

It is sometimes convenient to partition $\mathcal{F}$ into two disjoint sets: *defined functions* and *constructors.* For our purpose any function symbol that appears at the top of the left-hand side of a rule is defined, while all others are constructors. A term is said to be *flat* if it has at most one defined function with no function symbol nested below the defined function. Also, we will say that a rewrite system has a property (for example, left-linearity) if each of its rules has the said property.

Given a convergent rewrite system R, we denote an unification *goal* as $s \overset{?}{=} t$, for terms s and t. We say that the goal $s \overset{?}{=} t$ has a *solution* σ if $s\sigma \to^! w, t\sigma \to^! w$, for some term w. Similarly, a matching goal is written as $s \to^? t$ (where t is a normal form) and has a solution σ if $s\sigma \to^! t$. Furthermore, since we are interested in convergent systems alone, we use the notation $s \to^? x, t \to^? x$ to stand for an unification goal $s \overset{?}{=} t$ (x is a new variable, not in either s or t); this pair of goals has a solution σ if $s\sigma \to^! x\sigma, t\sigma \to^! x\sigma$.

For convergent systems, in general, semantic matching is as difficult as unification (for example, solving the goal $s \overset{?}{=} t$ in a convergent theory R is equivalent to solving the goal $eq(s,t) \to^? true$ in the theory $R \cup eq(x,x) \to true$). For *non-erasing* and *left-linear* rewrite systems matching is simpler than unification, as we will show in Section 3. However, these restrictions themselves do not suffice for decidable matching in such theories, as we will discuss in Section 4. In Section 4 we will introduce additional restrictions on the rewrite system such that matching becomes decidable, and show that each additional restriction that we put is necessary (i.e., matching becomes undecidable when we drop any of the additional restrictions). We start in

Section 2 by enumerating positive and negative results where the systems under consideration could be linear and/or flat.

2 Unification in Linear or Flat Systems

In this section we discuss matching and unification in convergent systems that additionally have restrictions of linearity or flatness (or both). Linearity and flatness are simple syntactic restrictions on the rewrite rules; it is easy to check if a given rewrite system has these properties. Such restrictions have been used extensively in the study of properties such as modular termination and confluence of rewrite systems. In this section, we use these restrictions to characterize decidability of matching and unification problems with convergent rewrite systems. The known results (including the new ones presented in this section) are summarized in Table 1. Some explanations are in order:

Restriction	**Decidable**	**Some References**
None	no	[Bockmayr, 1987] and others
Linear	no	[Heilbrunner and Hölldobler, 1987]
Left-linear	no	[Dershowitz and Jouannaud, 1990], also as an outcome of [Heilbrunner and Hölldobler, 1987]
Right-linear	no	as an outcome of [Heilbrunner and Hölldobler, 1987]
Left-flat	yes	[Christian, 1992]
Right-flat	no	Theorems 1 and 2
Flat	yes	as an outcome of [Christian, 1992]

Table 1: Results on unification and matching

- Each row of the table defines a new class of rewrite systems, by presenting the additional restrictions, over and above convergence.

- For each of the cases listed above, both matching and unification problems have the same property (i.e., either they are both decidable or both undecidable).

- The only positive result is the one from Christian [1992]. However, rewrite systems with flat left-hand sides are truly restrictive, and do not allow any recursively defined functions.

- Although linearity and flat right-hand sides do not guarantee decidability (as illustrated in the proof of Theorem 2), a further restriction does [Dershowitz and Mitra, 1993]. The additional restriction is to allow only one rule per defined function which has a right-hand side that is flat and contains exactly one defined function therein (i.e., for each defined function, there could be many rules with constructors and variables in the right-hand sides, but only one with a defined function on the right-hand side).

- In the results listed in Table 1 we have used flatness and linearity as additional restrictions on convergent systems. It is possible to combine the two requirements: For left-flat systems there is no requirement on linearity of any kind in the proof given in [Christian, 1992]; thus, the problems are decidable for left-flat non-linear systems. On the contrary, the counterexample in Theorem 2 is of a linear system with flat right-hand sides. Therefore, even for linear right-flat systems, the matching and unification problems continue to be undecidable.

In studying flat-systems, we started with the following result:

Theorem 1. *There is no decision procedure for the unification or matching problems in a convergent rewrite system, in which every right-hand side is flat.*

Proof. We show that the undecidable problem of emptyness of the intersection of two *simple* context-free languages can be encoded as a matching goal using a rewrite system with the given restrictions. See [Heilbrunner and Hölldobler, 1987] for the definition of a simple context free language, and a construction which motivated the following example:

Example 1. Consider the simple grammar in Greibach Normal Form (GNF) $\mathcal{G}$:

$$\mathcal{G} \equiv \{G \Rightarrow aBC, B \Rightarrow b, C \Rightarrow c\},$$

and the convergent rewrite system (the rewrite system has one defined function f; *cons*, $\cdot$, each terminal, non-terminal and the end symbol, \$, of $\mathcal{G}$ are constructors):

$$\begin{aligned} f(G\cdot x, cons(x', x''), a\cdot z) &\rightarrow f_G(x', x', x, x'', z) && (1)\\ f_G(B\cdot(C\cdot x), x', x, y, z) &\rightarrow f(x', y, z) && (2)\\ f(B\cdot x, cons(\$, y), b\cdot z) &\rightarrow f(x, y, z) && (3)\\ f(C\cdot x, cons(\$, y), c\cdot z) &\rightarrow f(x, y, z) && (4)\end{aligned}$$

For the construction of f, we have

$$f(G \cdot \$, x, z) \overset{?}{\to} f(\$, \$, \$) \text{ if and only if } z \in \mathcal{G}.$$

In general, given two simple context-free grammars $\mathcal{F}$ and $\mathcal{G}$, both in GNF (with start symbols F and G, respectively), we can construct a rewrite system with two defined functions, say f and g, such that f simulates a top-down parser for $\mathcal{F}$, and g for $\mathcal{G}$. The construction is similar to the one given for Example 1 above; we have to use two rules (similar to Rules 1 and 2) to simulate each production of the grammar (except when the production takes a non-terminal to a terminal string, in which case a single rule, such as Rule 3, would suffice). We need two rules for any general production (unlike [Heilbrunner and Hölldobler, 1987]) in order to ensure that the right-hand sides of rules are flat. With this construction, a goal of the form

$$c(f(F \cdot \$, x, z), g(G \cdot \$, x', z)) \overset{?}{\to} c(f(\$, \$, \$), g(\$, \$, \$))$$

(where c is a constructor) has a solution if and only if z is a string in the intersection of the two grammars, i.e,

$$c(f(F{\cdot}\$, x, z), g(G{\cdot}\$, x', z)) \overset{?}{\to} c(f(\$, \$, \$), g(\$, \$, \$)) \text{ if and only if } z \in \mathcal{F} \text{ and } z \in \mathcal{G}.$$

Since it is undecidable if the intersection of two simple context free languages is empty [Heilbrunner and Hölldobler, 1987], matching (and therefore unification) in such a rewrite system is also undecidable.

The proof of termination and confluence of the rewrite system with defined functions f and g can be done based on observations about production rules of a simple grammar in GNF. □

In the construction of Theorem 1 we had to use non-linear rules to get undecidability. However, even for strictly linear systems, the matching and unification problems continue to be undecidable:

Theorem 2. *There is no decision procedure for the unification or matching problem in a (left- and right-) linear convergent rewrite system, in which every right-hand side is flat.*

Proof. We show that semantic unification in such theories can be used to simulate the undecidable Post's Correspondence Problem (PCP, [Hopcroft and Ullman, 1979]).

An instance of PCP consists of two lists $A = w_1, \ldots, w_k$ and $B = x_1, \ldots, x_k$, of strings over some alphabet Σ. This instance has a *solution* if there exists a sequence of integers $i_1, \ldots, i_m, m \geq 1$, such that

$$w_{i_1}, \ldots, w_{i_m} = x_{i_1}, \ldots, x_{i_m}.$$

The following example illustrates how semantic unification can be used to generate solutions to a particular instance of the Post's Correspondence Problem:

Example 2. Let $\Sigma = \{s, p\}$, while A and B are as given below:

	List A	List B
i	w_i	x_i
1	s	sss
2	$spsss$	sp
3	sp	p

We construct the following convergent rewrite system R:

$$\begin{array}{rcl}
eq(s(x), s(y)) & \rightarrow & eq(x, y) \\
eq(p(x), p(y)) & \rightarrow & eq(x, y) \\
firsts(nil) & \rightarrow & nil \\
firsts(1 \cdot x) & \rightarrow & s(firsts(x)) \\
firsts(2 \cdot x) & \rightarrow & s(p(s(s(s(firsts(x)))))) \\
firsts(3 \cdot x) & \rightarrow & s(p(firsts(x))) \\
snds(nil) & \rightarrow & nil \\
snds(1 \cdot y) & \rightarrow & s(s(s(snds(y)))) \\
snds(2 \cdot y) & \rightarrow & s(p(snds(y))) \\
snds(3 \cdot y) & \rightarrow & p(snds(y))
\end{array}$$

It is easy to see that this instance of PCP has a solution if and only if the matching goal

$$eq(firsts(x \cdot y), snds(x \cdot y)) \overset{?}{\rightarrow} eq(nil, nil)$$

is satisfiable.

¿From the construction, it is evident that, given any instance of PCP, we can similarly construct a convergent rewrite system with the required syntactic restrictions, such that the matching problem described above has a solution if and only if the instance of PCP under consideration has one. The proof of convergence of the resulting rewrite system is based on the fact that no two left-hand sides unify, and that the system is terminating which can be shown using the recursive path ordering with the following precedence: $firsts > snds > s > p > nil$. Therefore, a decision procedure for matching (and thus unification) in such rewrite systems could be used to decide the Post's Correspondence Problem, which is impossible [Hopcroft and Ullman, 1979]. □

3 A Complete Matching Procedure

For convergent systems, the unification (and therefore matching) problem is recursively enumerable. In other words, there exists a procedure that can find a unifier (match) whenever one exists. Such unification (matching) procedures have been studied extensively in the literature; see, for example [Fay, 1979; Hullot, 1980; Hölldobler, 1987; Dershowitz and Sivakumar, 1987; Martelli *et al.*, 1989; Gallier and Snyder, 1990; Dershowitz *et al.*, 1990], and [Jouannaud and Kirchner, 1991], which is a survey of unification.

If we restrict ourselves to convergent rewrite systems that are, additionally, either non-erasing or left-linear, then the non-deterministic transformation rules of Table 2 constitutes a complete set for the matching problem:

Theorem 3 (Completeness). *Let $\mathcal{R}$ be either a left-linear or a non-erasing convergent rewrite system. If the goal $s \to^{?} N$ has a solution θ (that is, $s\theta \to^{!} N$), then there is a derivation of the form*

$$\{s \xrightarrow{?} N\} \overset{!}{\rightsquigarrow} \mu,$$

such that μ is a substitution at least as general as θ.

Proof. See Appendix. □

In fact, for non-erasing systems, Bind can be further simplified, as shown in [Mitra, 1994]. The system of Table 2, however, is incomplete for general unification: For example, if we have the goals $s \to^{?} x, t \to^{?} x$, for terms s and t and variable x, such that both s and t have a constructor at the root position, then none of the transformation rules apply (Eliminate or Bind doesn't apply, since s and t are non-variable terms; furthermore, we cannot mutate, since the root symbols are constructors, and we cannot decompose since the

Eliminate	$\{x \rightarrow^? t\}$ $\leadsto$ $\{x \mapsto t\}$ where x is a free-variable that does not occur in t
Bind	$\{x \rightarrow^? s, x \mapsto t\}$ $\leadsto$ $x \mapsto s, mgu(s,t)$ if x does not occur in s
Mutate	$\{f(s_1, \ldots, s_n) \rightarrow^? t\}$ $\leadsto$ $\{s_1 \rightarrow^? l_1, \ldots, s_n \rightarrow^? l_n, r \rightarrow^? t\}$ where $f(l_1, \ldots, l_n) \rightarrow r$ is a renamed rule in R
Decompose	$\{f(s_1, \ldots, s_n) \rightarrow^? f(t_1, \ldots, t_n)\}$ $\leadsto$ $\{s_1 \rightarrow^? t_1, \ldots, s_n \rightarrow^? t_n\}$

Table 2: Transformation rules for semantic matching with left-linear or non-erasing convergent systems

right-hand sides are variables). However, if we augment the transformation rules of Table 2 with those of Table 3, the resulting system is sufficient for generating a complete set of unifiers in theories defined by convergent rewrite systems [Mitra, 1994]. In fact, Imitate is the only new transformation rule that we need, since Apply is used for matching whenever bindings take place (see the proof of Theorem 3). In this sense, matching is simpler than unification for such restricted convergent systems.

4 Decidable Matching

For convergent systems, in general, semantic matching is as difficult as semantic unification. For example, solving the goal $s \overset{?}{=} t$ in a convergent theory R is equivalent to solving the goal $eq(s,t) \rightarrow^? true$ in the theory $R \cup eq(x,x) \rightarrow true$, for a new function symbol eq and constant $true$; the augmented theory is convergent since eq is a new symbol, not in R. However, in Theorem 3 we have identified two classes of (those of left-linear and non-erasing) systems for which matching is simpler than unification. Therefore, one natural question is: Under what conditions is matching decidable (independent of unification) for such classes of systems? Unfortunately, as listed

Imitate	$f(s_1, \ldots, s_n) \rightarrow^? x$ $\rightsquigarrow$ $s_1 \rightarrow^? x_1, \ldots, s_n \rightarrow^? x_n, x \mapsto f(x_1, \ldots, x_n)$ where x is an unbound variable, and $x_1, \ldots, x_n$ are new variables
Apply	$s \rightarrow^? t, \sigma$ $\rightsquigarrow$ $s \rightarrow^? t\sigma, \sigma$

Table 3: Additional transformation rules for semantic unification

in Table 1, left-linearity alone is not enough to ensure decidability of matching; in fact, the rewrite system constructed to simulate PCP in Example 2 is linear and non-erasing; therefore, neither linearity, nor non-erasing (nor a combination of the two) is a strong enough criterion to guarantee decidability of matching. On the positive side, Proposition 4 identifies one class of rewrite systems (with additional restrictions, over and above non-erasing) for which the matching problem is indeed decidable:

Definition 1 (Non-Decreasing). A function symbol f is defined to be *non-decreasing* (with respect to depth) if whenever $f(s_1, \ldots, s_n) \rightarrow^! N$, where each s_i and N are ground normal forms, $depth(s_i) \leq depth(N)$. Any function which does not have this property is said to be a (*potentially*) *decreasing* function.

Proposition 4 ([Dershowitz *et al.*, 1992]). *Let $\mathcal{R}$ be a convergent non-erasing rewrite system. If all right-hand sides for rules in $\mathcal{R}$ are either variables, or have a constructor at the root, and all right-hand sides are such that no defined function is nested below any decreasing function, then the semantic matching problem is decidable for $\mathcal{R}$.*

Furthermore, it was shown, in [Dershowitz *et al.*, 1992], that each of the restrictions listed in Proposition 4 is necessary to have decidable matching. In the same spirit, we have Theorem 5 for left-linear systems. For this theorem, we will assume that constants and variables have depth one:

Theorem 5. *Let $\mathcal{R}$ be a convergent left-linear rewrite system. If for every rule $f(l_1, \ldots, l_n) \rightarrow r$ in $\mathcal{R}$*

1. *each $l_i, 1 \leq i \leq n$, is of depth at most two,*

2. *r is either a variable or has a constructor at the root, and*

3. *whenever at least one l_j has depth greater than one, r has depth greater than one,*

then the semantic matching problem is decidable for $\mathcal{R}$.

Proof. See Appendix. □

Example 3. The following definition of squaring using $+$ and $*$ obeys all the syntactic restrictions of Theorem 5, and therefore has a decidable matching problem:

$$
\begin{array}{rcl}
0 + x & \rightarrow & x \\
s(x) + y & \rightarrow & s(x + y) \\[4pt]
0 * x & \rightarrow & 0 \\
x * 0 & \rightarrow & 0 \\
s(x) * s(y) & \rightarrow & s(y + (x * s(y))) \\[4pt]
sq(0) & \rightarrow & 0 \\
sq(s(x)) & \rightarrow & s(sq(x) + (s(s(0)) * x))
\end{array}
$$

Example 4. As another example, consider inserting a number in its correct place, in a list of numbers:

$$
\begin{array}{rcl}
\mathit{min}(x, 0) & \rightarrow & 0 \\
\mathit{min}(0, x) & \rightarrow & 0 \\
\mathit{min}(s(x), s(y)) & \rightarrow & s(\mathit{min}(x, y)) \\[4pt]
\mathit{max}(x, 0) & \rightarrow & x \\
\mathit{max}(0, x) & \rightarrow & x \\
\mathit{max}(s(x), s(y)) & \rightarrow & s(\mathit{max}(x, y)) \\[4pt]
\mathit{insert}(x, \mathit{nil}) & \rightarrow & x \cdot \mathit{nil} \\
\mathit{insert}(x, y \cdot z) & \rightarrow & \mathit{min}(x, y) \cdot \mathit{insert}(\mathit{max}(x, y), z)
\end{array}
$$

We now show that each of the restrictions used in Theorem 5 is necessary for decidability: If we drop the requirement of left-linearity, then we could get undecidability, essentially, by encoding the more general unification problem in left-linear systems as a matching problem (see Section 4 of [Dershowitz *et al.*, 1992] for examples). In the remaining cases, we show that matching of certain goals would result in unification in the theories of addition ($+$) and multiplication ($*$). (Notice that the definitions of $+$ and $*$ in Example 3 obey all the syntactic restrictions of Theorem 5. Thus, the matching problem is decidable for this system. However, the unification problem is undecidable, due to the undecidability of the Hilbert's Tenth Problem.) First of all, we relax Condition 3, that is, we allow subterms of depth greater than one below the root on the left-hand side, without requiring that the right-hand side be of depth at least two. The following example illustrates the problem:

Example 5. Consider the rewrite system consisting of the rules for addition and multiplication (first 5 rules from Example 3) together with the following rewrite rules (the collective system can be proved to be convergent):

$$\begin{aligned} f(1) &\rightarrow 1 && (5)\\ f(s(1)) &\rightarrow 1 && (6)\end{aligned}$$

$$\begin{aligned} g(1,1) &\rightarrow s(1) && (7)\\ g(s(x),s(y)) &\rightarrow s(f(g(x,y))) && (8)\end{aligned}$$

Rule 6 is the only one which violates the nesting criterion.

Lemma 6. *Given the rewrite system of Example 5, we have*

$$g(x,y) = s(1) \textit{ if and only if } x = y = s^n(1), n \geq 0,$$

where $s^0(1) = 1, s^{n+1}(1) = s(s^n(1))$.

Proof. Let $>$ be the ordering on terms such that $s > t$ if either $s \rightarrow t$ or t is a proper subterm of s. Since $\rightarrow$ is well-founded, so is $>$. We prove the lemma by induction on the ordering $>$:

Base When $n = 0$, we have $g(1,1) \rightarrow s(1)$. Therefore, the proposition holds for the base case.

Induction We now establish the proposition for $n+1$:

$$g(s^{n+1}(1), s^{n+1}(1)) \equiv g(s(s^n(1)), s(s^n(1))) \rightarrow s(f(g(s^n(1), s^n(1)))).$$

Therefore, using the inductive hypothesis (since $g(s^{n+1}(1), s^{n+1}(1)) > g(s^n(1), s^n(1))$), we get:

$$g(s^{n+1}(1), s^{n+1}(1)) \rightarrow^* s(f(g(s^n(1), s^n(1)))) = s(f(s(1))) \rightarrow s(1),$$

which completes the proof (we have used $\equiv$ to denote syntactic equality, while $=$ has been used for the inductive step).

□

Theorem 7. *The matching problem is undecidable for the rewrite system of Example 5.*

Proof. Suppose t and t' are general terms involving $+$ and $*$ alone. Therefore, by Lemma 6, a goal of the form $g(t', t) \rightarrow^{?} s(1)$, would, in general, be undecidable, since a decision procedure for this problem could be used to solve the Hilbert's Tenth Problem, which is impossible. □

Thereafter, we relax Condition 2, by allowing defined functions to appear as the root of the right-hand sides of rules. We get the following variant of Example 5:

Example 6. As in Example 5, consider the convergent rewrite system consisting of the rules for addition and multiplication, together with the following additional rewrite rules:

$$f(1) \quad \rightarrow \quad 1 \tag{9}$$

$$g(1,1) \quad \rightarrow \quad 1 \tag{10}$$

$$g(s(x), s(y)) \quad \rightarrow \quad f(g(x,y)) \tag{11}$$

Notice that Rule 11 is the only one which violates Condition 2. Therefore, we have the following:

Lemma 8. *Given the rewrite system of Example 6, we have:*

$$g(x,y) = 1 \textit{ if and only if } x = y = s^n(1), n \geq 0.$$

Theorem 9. *The matching problem is undecidable for the rewrite system of Example 6.*

Finally, we relax Condition 1, and allow depths greater than two below the root function on left-hand sides of rules (but in order to make sure that the last condition not be violated, we would insist that whatever depth we have on the left-hand side must show up on every path on the right-hand side, by way of leading constructors). Consider the following example, wherein, we encode f from Example 5 using new rules:

Example 7. Consider the convergent rewrite system consisting of the rules for addition and multiplication, together with the following additional rewrite rules:

$$F(s(x)) \rightarrow s(1) \tag{12}$$

$$G(s(s(1)), s(s(x))) \rightarrow s(s(s(x))) \tag{13}$$

$$g(1, 1) \rightarrow s(s(1)) \tag{14}$$

$$g(s(x), s(y)) \rightarrow s(F(G(g(x, y), s(s(1))))) \tag{15}$$

Notice that none of the rules have an immediate subterm on the left-hand side that is of greater depth than the depth of the corresponding right-hand side. However, Rule 12 erases x; while Rule 13 is the only one which violates the depth criterion for left-hand sides (it allows immediate subterms of depth 3 on its left-hand side).

For this rewrite system, we have:

$$F(G(s(s(1)), s(s(x)))) \rightarrow F(s(s(s(x)))) \rightarrow^{!} s(1),$$

where x is a variable. Therefore, like in the previous cases, we have:

Lemma 10. *Given the rewrite system of Example 7, we have:*

$$g(x, y) = s(s(1)) \textit{ if and only if } x = y = s^n(1), n \geq 0.$$

Theorem 11. *The matching problem is undecidable for the rewrite system of Example 7.*

5 Conclusion

Semantic unification and matching are useful in incorporating logic-programming capabilities in a functional language, in constraint based systems and in theorem proving. Even when a system admits a convergent presentation, the corresponding unification or matching procedure may be undecidable.

In this paper we have studied restricted convergent systems for which matching is simpler than unification. In particular, we have provided a positive result for a special class of left-linear convergent rewrite systems for which the matching problem is decidable. In fact, we have shown, by way of counterexamples, that matching becomes undecidable (as it usually is) when any of the conditions that we have proposed is weakened. The result is similar in spirit to one given in [Dershowitz *et al.*, 1992] which was for restricted non-erasing systems. Since we have shown that these two classes (i.e., left-linear and non-erasing) are the ones for which matching is simpler than unification, the current result completes the characterization of decidable matching by providing the dual of the result from [Dershowitz *et al.*, 1992]. One difference between the two characterizations is that the current theorem uses the syntactic property of depth, while the one from [Dershowitz *et al.*, 1992] used a semantic property of decreasing functions. Recently [Aguzzi and Modigliani, 1994] have extended the decidability criterion of [Dershowitz *et al.*, 1992] by introducing the notion of *P-increase* (*P-non-decrease*) to replace increase (non-decrease), using which one could talk about the positional increase (non-decrease) of a suitable property like depth, with recursive calls (rather than use the property on an entire function). By using positional information, it is also possible to handle certain rewrite systems which does not have leading constructors on the right-hand sides of rules (for example, the usual presentation of $*$ contains rules $\{x * 0 \rightarrow 0, s(x) * y \rightarrow y + (x * y)\}$, instead of the 3 rules of Example 3). We believe that a similar refinement (of using positional depth, rather than depth of the whole term) is possible for the new result on semantic matching presented in this paper. In fact, with this extension, we should be able to handle *insertion-sort* by adding the following rules to those of Example 4:

$$sort(nil) \rightarrow nil, \quad sort(x \cdot y) \rightarrow insert(x, sort(y)).$$

Linearity and flatness of rewrite rules are common syntactic properties which have been used for many different characterizations of rewrite systems (for example, confluence and termination). In this paper we have shown how these criteria affect unification and matching in convergent systems. In most cases the results turn out to be negative, as was the case with the new result of this paper. However, with this new information, we have been able to complete the characterization of unification and matching for systems with these two syntactic restrictions. Comon et al. [1991], use the idea of proving termination of a system of transformation rules for characterizing decidable unification, but with no assumption of convergence. In their case, both sides of the given equations must satisfy a slightly different non-nesting requirement than that of flatness as described here. (Since equations can be used in either direction, it is intuitive to use the restriction on both sides.) The resulting theories are simpler than the ones for convergent systems with flat right-hand sides (in fact, they bear similarity to the systems that are convergent and left-flat, for which a positive result was proved in [Christian, 1992]). Other characterizations of decidable unification appear in [Kapur and Narendran, 1987] (every right-hand side of a rule must be a proper subterm of the corresponding left-hand side) and [Hullot, 1980] (every right-hand side must be a variable or a ground term). Unfortunately, none of these systems are powerful enough to capture truly recursive functions. Decidability results for unification in convergent systems was extended in [Dershowitz and Mitra, 1993], where the problems considered could potentially have infinite solutions, which were captured as indexed terms, along the lines of [Comon, 1992]. The main difference between the requirements of the result from [Dershowitz and Mitra, 1993] and the one presented here is that we allow multiple flat terms with defined functions as right-hand sides for rules defining a given function. For instance, the definition of *eq*, in Example 2, used two rules, each of which has *eq* (a defined function) on its right-hand side, which would not be allowed by [Dershowitz and Mitra, 1993].

References

[Aguzzi and Modigliani, 1994] G. Aguzzi and U. Modigliani. A Criterion to Decide the Semantic Matching Problem. In *Proceedings of the International Conference on Logic and Algebra (in memory of Prof. Magari)*, Italy, 1995.

[Bockmayr, 1987] A. Bockmayr. A Note on a Canonical Theory with Undecidable Unification and Matching Problem. *Journal of Automated Reasoning*, Vol 3, pages 379–381, 1987.

[Christian, 1992] J. Christian. Some Termination Criteria for Narrowing and E-Narrowing. In *Proceeding of the Eleventh International Conference on Automated Deduction*, Saratoga Springs, New York, June 1992. Volume 607, pages 582–588, of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

[Comon, 1992] H. Comon. On unification of terms with integer exponents. Technical Report 770, Universite de Paris-Sud, Laboratoire de Recherche en Informatique, 1992.

[Comon *et al.*, 1991] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Decidable problems in shallow equational theories. Technical Report 718, Universite de Paris-Sud, Laboratoire de Recherche en Informatique, December 1991.

[Dershowitz and Sivakumar, 1987] N. Dershowitz and G. Sivakumar. Solving goals in equational languages. In S. Kaplan and J.-P. Jouannaud, editors, *Proceedings of the First International Workshop on Conditional Term Rewriting Systems*, pages 45–55, Orsay, France, July 1987. Vol. 308 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (1988).

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320, North-Holland, Amsterdam, 1990.

[Dershowitz *et al.*, 1990] N. Dershowitz, S. Mitra and G. Sivakumar. Equation solving in conditional AC-theories. In *Proceedings of the Second International Conference on Algebraic and Logic Programming*, Nancy, France, 1990. Volume 463, pages 283–297, of *Lecture Notes in Computer Science*, Springer Verlag.

[Dershowitz *et al.*, 1992] N. Dershowitz, S. Mitra, and G. Sivakumar. Decidable matching for convergent systems. In *Proceedings of the Eleventh Conference on Automated Deduction*, pages 589–602, Saratoga Springs, NY, June 1992. Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin.

[Dershowitz and Mitra, 1993] N. Dershowitz and S. Mitra. Higher-order and semantic unification. In *Proceedings of the Thirteenth International Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India, 1993. Volume 761, pages 139–150, of *Lecture Notes in Computer Science*, Springer Verlag.

[Fay, 1979] M. Fay. First-order unification in an equational theory. In *Proceedings of the Fourth Workshop on Automated Deduction*, pages 161–167, Austin, TX, February 1979.

[Gallier and Snyder, 1990] J. Gallier and W. Snyder. Complete set of transformations for general E-unification. *Theoretical Computer Science*, 67:203–260, 1990.

[Heilbrunner and Hölldobler, 1987] S. Heilbrunner and S. Hölldobler. The undecidability of the unification and matching problem for canonical theories. *Acta Informatica*, 24(2):157–171, April 1987.

[Hölldobler, 1987] S. Hölldobler. A unification algorithm for confluent theories. In *Proceedings of the Fourteenth EATCS International Colloquium on Automata, Languages, and Programming*, pages 31–41, Kalsruhe, West Germany, July 1987. Vol. 267 in *Lecture Notes in Computer Science*, Springer, Berlin.

[Hopcroft and Ullman, 1979] J. E. Hopcroft and J. D. Ullman. Introduction to automata theory, languages and computation. Addison Wesley, 1979.

[Hullot, 1980] J.-M. Hullot. Canonical forms and unification. In R. Kowalski, editor, *Proceedings of the Fifth International Conference on Automated Deduction*, pages 318–334, Les Arcs, France, July 1980. Vol. 87 of *Lecture Notes in Computer Science*, Springer, Berlin.

[Jouannaud and Kirchner, 1991] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, Cambridge, MA, 1991.

[Kapur and Narendran, 1987] D. Kapur and P. Narendran. Matching, unification and complexity (a preliminary note). *SIGSAM Bulletin*, 21(4):6–9, November 1987.

[Martelli *et al.*, 1989] A. Martelli, G. F. Rossi, and C. Moiso. Lazy unification algorithms for canonical rewrite systems. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2: Rewriting Techniques, pages 245–274. Academic Press, New York, 1989.

[Mitra, 1994] S. Mitra. Semantic Unification for Convergent Systems. PhD thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1994. Appears as Dept. of Computer Science Technical Report Number UIUCDCS-R-94-1855, October, 1994.

6 Appendix: Proofs

6.1 Completeness of Matching

We start with Theorem 3, which we prove in two parts, Theorems 12 (for non-erasing systems) and 13 (for left-linear systems).

Theorem 12 (Completeness). *Let $\mathcal{R}$ be a non-erasing convergent rewrite system. If the goal $s \to^? N$ has a solution θ (that is, $s\theta \to^! N$), then there is a derivation of the form*

$$\{s \overset{?}{\to} N\} \overset{!}{\rightsquigarrow} \mu,$$

such that μ is a substitution at least as general as θ.

The proof is similar to one given for the Completeness of unification (i.e., of transformation rules from Tables 2 and 3) given in [Mitra, 1994]. The only difference is that in this proof we have to use a selection strategy to select the next goal from a collection of yet unsolved goals, whereas in the proof of [Mitra, 1994], the choice could be non-deterministic. We start with a few definitions:

Definition 2 (Node). Let $G \equiv \{s_1 \to^? t_1, \ldots, s_n \to^? t_n\}$ be a set of goals and $\sigma \equiv \{x_1 \mapsto u_1, \ldots, x_m \mapsto u_m\}$ be a substitution. Then the collection

$$\{s_1 \overset{?}{\to} t_1, \ldots, s_n \overset{?}{\to} t_n, x_1 \mapsto u_1, \ldots, x_m \mapsto u_m\}$$

is called a *node*.

With G and σ as defined above, we write nodes as $\langle G; \sigma \rangle$.

Definition 3 (Ordering $\succ_\theta$). Let $>$ be the smallest ordering on terms such that $s > t$ if and only if $s \to t$ or t is a proper subterm of s. We define the ordering $\succ_\theta$ on nodes, with respect to a solution θ: Let $\langle G_1; \tau_1 \rangle$ and $\langle G_2; \tau_2 \rangle$ be two nodes, each of which admit a solution θ; then, $\langle G_1; \tau_1 \rangle \succ_\theta \langle G_2; \tau_2 \rangle$ if and only if $\{s_1\theta, \ldots, s_n\theta\} \succ \{u_1\theta, \ldots, u_n\theta\}$, where $G_1 \equiv \{s_1 \to^? t_1, \ldots, s_n \to^? t_n\}$, $G_2 \equiv \{u_1 \to^? v_1, \ldots, u_m \to^? v_m\}$ and $\succ$ is the multiset extension of $>$.

Proof. (*Sketch*) In the statement of Theorem 12 we have started with a single matching goal of the form $s \to^? N$. However, in general, we will have a collection of goals to be solved, together with a substitution (which is the partial computed solution).

Given that we start with a goal of the form $s \rightarrow^? N$ (that has a ground term N as the right-hand side) it suffices to only consider goals of this form: If we were to decompose this starting goal, then all the subgoals would be of the same form, and therefore the proposition holds. However, if we were to mutate $s \equiv f(s_1, \ldots, s_n) \rightarrow^? N$, using the rule $f(l_1, \ldots, l_n) \rightarrow r$ then we would have a collection of subgoals of the form

$$s_1 \overset{?}{\rightarrow} l_1, \ldots, s_n \overset{?}{\rightarrow} l_n, r \overset{?}{\rightarrow} N.$$

Thereafter, we have a subgoal $r \rightarrow^? N$ which has the required property, and can be solved next. Suppose $r \rightarrow^? N$ produces a solution σ. Then, each variable of r must have been bound to a ground term in σ (if not, we would have the situation that a non-ground term $r\sigma$ rewrites to a ground normal form N, using non-erasing rules alone, which is impossible). Furthermore, by the requirements of the theorem, every variable in $f(l_1, \ldots, l_n)$ is also in r. Thus, using the previous two observations, we have that $l_i\sigma, 1 \leq i \leq n$, must be ground. At this point, it suffices to solve any of the goals from

$$s_1 \overset{?}{\rightarrow} l_1\sigma, \ldots, s_n \overset{?}{\rightarrow} l_n\sigma,$$

each of which has a ground right-hand side.

Therefore, we are going to use the following selection strategy for solving goals: whenever we use Mutate on a goal $s \rightarrow^? t$, we will solve the $r \rightarrow^? t$ subgoal first to get a solution; we apply the solution to the right-hand sides of the remaining subgoals, and then select one of $s_1 \rightarrow^? l_1, \ldots, s_n \rightarrow^? l_n$. At other times, we will always have a subgoal with a ground right-hand side, which we select (if there are several such subgoals, we do not care as to which one is picked).

The rest of the proof is as follows: Once we pick a subgoal (say $s \rightarrow^? t$) using the selection strategy above, we apply the non-deterministic transformation rules from Table 2. For example, if $s \equiv x$, x being a variable, then one of Eliminate or Bind would apply. In either case, the new collection of goals can be shown to be smaller (in $\succ_\theta$) than the original one. Furthermore, each such application is *solution preserving* (i.e., any solution to the collection before application of the transformation rule, is also a solution to the collection generated by applying the transformation). Of course, if $s \equiv f(s_1, \ldots, s_n)$ (i.e., s is not a variable) then it may be possible to use Decompose and Mutate on this goal (in fact, it may be possible to Mutate with different rules from $\mathcal{R}$); for completeness, each such possibility has to be attempted. In this case, we could demonstrate similar properties, i.e., that the generated goals are smaller (in $\succ_\theta$) than the original ones, and that some solution preserving transformation rule applies. Details are the same as in the proof of unification given in [Mitra, 1994]. □

Theorem 13 (Completeness). *Let $\mathcal{R}$ be a left-linear convergent rewrite system. If the goal $s \rightarrow^? N$ has a solution θ (that is, $s\theta \rightarrow^! N$), then there is a derivation of the form*

$$\{s \overset{?}{\rightarrow} N\} \overset{!}{\rightsquigarrow} \mu,$$

such that μ is a substitution at least as general as θ.

We would continue to use the same selection strategy, as mentioned before, for proving completeness for left-linear rewrite systems. We need the following lemmata for the proof:

Lemma 14. *Let $\mathcal{R}$ be a left-linear convergent rewrite system. Then, for the initial goal $\{s \rightarrow^? N\}$, where N is ground, if $G \cup \{t \rightarrow^? t'\}$ is the set of subgoals generated by the procedure at some point and x is a variable in t', then t' must be linear with respect to x; furthermore, x does not occur in any right-hand side of a subgoal in G.*

Proof. If $\tau \rightarrow^? t$ is a subgoal generated by the procedure for the initial goal $s \rightarrow^? N$, then the variables of t must come either from N or from the left-hand side of some rule in $\mathcal{R}$. For our case, N is ground, and $\mathcal{R}$ is left-linear, thus this variable cannot occur in any other right-hand side in the goal set, and again t itself must be linear in this variable. □

Lemma 15. *Let $\mathcal{R}$ be a left-linear convergent rewrite system. Then, in solving $\{t \rightarrow^? N\}$, a subgoal of the form $s \rightarrow^? x$ can be ignored without having to solve it any further.*

Proof. Let G denote the remaining set of subgoals when $s \rightarrow^? x$ is encountered. By Lemma 14, x cannot appear in any right-hand side in G. Furthermore, we always solve subgoals of mutation using the following strategy (as was used in the proof of Theorem 12): solve the $r \rightarrow^? t$ subgoal first, to get a solution σ, and then solve the remaining subgoals $s_i \rightarrow^? l_i\sigma$ in any order. Therefore, if we ever encounter a goal of the form $s \rightarrow^? x$, then x must be a variable which does not require instantiation (that is, x does not appear anywhere in the remaining subgoals). Thus, any such goal is trivially solvable (that is, it has a solution for any substitution for the variables in s), and can be ignored. (As a result, all the variables in s would have indeterminate solutions, unless they appear on the left-hand side of some other goal which causes instantiation.) □

We now state a proof of the theorem 13:

Proof. Notice that the major difference between the transformation rules for semantic unification (i.e, the collective rules from Tables 2 and 3) and those for matching in left-linear systems (those from Table 2 alone) is that the latter does not have the rule for imitation. However, by Lemma 15, whenever the right-hand side of a goal is a variable, that goal is trivially solvable. Therefore, the remaining transformation rules are enough to enumerate a complete set of solutions of any matching goal in this case (given that the collective system from Tables 2 and 3 is complete for semantic unification in convergent rewrite systems [Mitra, 1994]).

Note that, as in the case of non-erasing systems, we could apply partial solutions as required, and therefore do not have to explicitly consider Apply. □

6.2 Decidable Matching

Since we are interested in left-linear systems, it is sufficient to consider only transformation rules from Table 2. Also, the proof of Theorem 3 uses a particular selection strategy for picking subgoals to solve (after mutation, we always solve the $r \rightarrow^? t$ subgoal first, before solving the $s_i \rightarrow^? l_i$ in any order; after decomposition we could solve the new subgoals in any order); we continue to use the same selection strategy for solving subgoals. We will use the following lemma:

Lemma 16. *The most general unifier computed in Bind need only deal with linear terms.*

Proof. We may have to use bind because the rewrite rules may have non-linear variables on its right-hand sides, and since the left-hand side of the starting goal may be non-linear.

Suppose we start with the goal $S \rightarrow^? N$, for a ground normal-form N, use the selection strategy mentioned before, and apply transformation rules. Furthermore, consider the sequence of transformation rules before the first application of Bind. In this sequence, the generated substitutions should have been produced using Eliminate alone. Therefore, each term bound to a variable in this substitution must be linear (such terms must have come from N or from the left-hand sides of applied rules). Therefore, both s and t of Bind must be linear, and of independent variables. Thus, the computed *mgu* would again be a linear term. Furthermore, any of the linear variables in either s or t that gets bound during the mgu computation can be removed, since they do not appear anywhere else in either goals or substitutions, due to left-linear rules and the selection strategy. □

We now state a proof of Theorem 5:

Proof. Since we have a left-linear system, it suffices to consider transformation rules from Table 2 alone, that is, we do not have to consider Apply and Imitate. Furthermore, due to left-linearity, we only need to solve goals of the form $s \rightarrow^? t$, where any variable x in t is linear in t and does not occur in the right-hand side of any other subgoal (this is true because we have directed goals, and terms on the right-hand sides of goals could either be left-hand sides of (left-linear) rules from previous mutations, or subterms of the ground term N, when solving for a initial goal of the form $s' \rightarrow^? N$; see Lemma 14).

Let $\succ$ be the well-founded ordering on goals such that $s_1 \rightarrow^? t_1 \succ s_2 \rightarrow^? t_2$ if either:

- $depth(t_1) > depth(t_2)$, or
- $depth(t_1) = depth(t_2)$ and s_2 is a proper subterm of s_1.

The proof proceeds by picking a subgoal (say $s \rightarrow^? t$) from the remaining ones (following the selection strategy mentioned before). We show by induction on the multiset extension of the ordering $\succ$ that any solution to this goal is bounded in depth by that of t, and every application of a transformation rule decreases the complexity of goals in this ordering.

If the goal is of the form $x \rightarrow^? t$ (x being a variable) then there are two cases, either Eliminate applies (in which case the proposition is trivially true, since the goal gets removed from the collection and a binding gets added, but our ordering does not consider bindings) or Bind applies. In the latter case, given Lemma 16, we need only compute the most general unifier of two linear terms with independent variables. Therefore, the computed mgu is linear; furthermore, the depth is bounded by that of the deeper of the two terms for which the mgu is being computed. For the other cases, let $s = f(s_1, \ldots, s_n)$ and $t = N[\bar{x}]$. (We use the notation $N[\bar{x}]$ to denote a term linear in variables $\bar{x}$ and for which no other subgoal in the current set has any of these variables on the right-hand side; Lemma 14 shows why considering such goals is sufficient.) There are several cases to be considered:

- If $N[\bar{x}]$ is a variable, then by Lemma 15, we do not have to solve this goal any further. Therefore, the only solution to this goal is an indeterminate (unbound variable) for each variable of $f(s_1, \ldots, s_n)$. Thus, the solution is of depth one, which is the same as that of $N[\bar{x}]$.

- If $N[\bar{x}]$ is a constant, then for decomposition to work, $f(s_1, \ldots, s_n)$ must be the identical constant, which gives the empty substitution as the only solution, and therefore the hypothesis holds in this case. Decrease in complexity is caused by the removal of the subgoal under consideration.

  Next, consider mutation of the goal $f(s_1, \ldots, s_n) \rightarrow^? N[\bar{x}]$, $N[\bar{x}]$ a constant, using a rule of the form $f(l_1, \ldots, l_n) \rightarrow r$. The only time such a rule could work is if $depth(r) = 1$. (For any other rule, by the assumption of the theorem, there has to be a constructor at the root of r, which would lead to failure when solving the $r \rightarrow^? N[\bar{x}]$ subgoal.) Furthermore, since r has depth one, by the assumption of the theorem, each $l_i, 1 \leq i \leq n$, must be of depth one also (Condition 3). If r is a constant (it also has to be a constructor, by Condition 2), then the only possible solution to the goal $r \rightarrow^? N[\bar{x}]$ is the empty substitution ($\sigma = \{\}$). However, if r is a variable, say z, then this goal has a unique solution of depth one (the solution is $\sigma = \{z \mapsto N[\bar{x}]\}$). Therefore, the derivation looks like:

  $$f(s_1, \ldots, s_n) \rightarrow^? N[\bar{x}] \quad \rightsquigarrow^* \quad s_1 \rightarrow^? l_1\sigma, \cdots, s_n \rightarrow^? l_n\sigma, \sigma$$

  In either case, each of the subgoals $s_1 \rightarrow^? l_1\sigma, \cdots, s_n \rightarrow^? l_n\sigma$ is smaller than the original goal $f(s_1, \ldots, s_n) \rightarrow^? N[\bar{x}]$ (since each l_i is either a variable or a constant, thus $depth(l_i\sigma) = 1$), and the proposition follows by induction on these smaller subgoals.

- For any other case, the depth of $N[\bar{x}]$ is at least two; let $N[\bar{x}] \equiv g(N_1, \ldots, N_m)$. Were we to decompose the goal, then each of the subgoals thus generated would be smaller in the ordering $\succ$. Thus, for decomposition, the proposition holds by induction on each of the smaller subgoals. Finally, consider mutation of this goal using a rule of the form $f(l_1, \ldots, l_n) \rightarrow r$:

  $$f(s_1, \ldots, s_n) \overset{?}{\rightarrow} N[\bar{x}] \rightsquigarrow_{\mathbf{Mutate}} s_1 \overset{?}{\rightarrow} l_1, \cdots, s_n \overset{?}{\rightarrow} l_n, r \overset{?}{\rightarrow} N[\bar{x}],$$

  there are further cases:

  - If we require r to be of depth one, then r must be a variable, say z. (A constant for r does not work, since the constant must be a constructor by the requirements of the theorem, and therefore, the goal $r \rightarrow^? N[\bar{x}]$ has no solution.) In this case the subgoal

$r \rightarrow^? N[\bar{x}]$ (that is, $z \rightarrow^? N[\bar{x}]$) is trivially solvable, and the solution is bounded in depth by that of $N[\bar{x}]$. Furthermore, by the assumption of the theorem, each $l_i, 1 \leq i \leq n$, has depth one (given Condition 3, since we assumed r to be of depth one). Suppose σ is the (unique) solution to the goal $z \rightarrow^? N[\bar{x}]$. Therefore, as in the previous case, we have $depth(l_i\sigma) \leq depth(N[\bar{x}]), 1 \leq i \leq n$. Thus, the proposition holds by applying the hypothesis on the smaller subgoals $s_1 \rightarrow^? l_1\sigma, \cdots, s_n \rightarrow^? l_n\sigma$.

- If r has depth greater than one, then it must have a leading constructor (by assumption of the theorem). Suppose $r \equiv g(r_1, \ldots, r_m)$, where g is a constructor (if the root function of r is different from g, then we get failure, so this is the only case to be considered). Thus, it is possible to decompose the goal $r \rightarrow^? N[\bar{x}]$ at least once, leading to smaller subgoals of the form $r_1 \rightarrow^? N_1, \ldots, r_m \rightarrow^? N_m$ (that is, $f(s_1, \ldots, s_n) \rightarrow^? N[\bar{x}] \succ r_i \rightarrow^? N_i, 1 \leq i \leq m$). Furthermore, let $depth(N[\bar{x}]) = d \geq 2$, which means that $max(depth(N_i)) = d - 1, 1 \leq i \leq m$. Although we could solve these subgoal in any order, for clarity of presentation, let us assume that we solve them in left-to-right sequence. In other words, we first solve $r_1 \rightarrow^? N_1$, to get a solution σ_1 (which, by inductive hypothesis, must be bounded in size by $d - 1$). Next, we solve $r_2 \rightarrow^? N_2\sigma_1$. Due to the linearity requirements on $N[\bar{x}]$, $N_2\sigma_1 \equiv N_2$, and therefore, we could still use the inductive hypothesis on this goal. Eventually, we would solve $r_m \rightarrow^? N_m\sigma_{m-1}$, where σ_{m-1} is the collective solution from solving $r_i \rightarrow^? N_i, 1 \leq i \leq m-1$, and we again apply the same technique, to get the solution σ_m, which again should be bounded by $d-1$. Notice that σ_m is indeed a solution to $r \rightarrow^? N[\bar{x}]$: therefore, we have demonstrated that any solution to $r \rightarrow^? N[\bar{x}]$ would be bounded in depth by $d - 1$. Let σ be such a solution, which gives us the new set of subgoals as:

$$f(s_1, \ldots, s_n) \rightarrow^? N[\bar{x}] \quad \leadsto^* \quad s_1 \rightarrow^? l_1\sigma, \cdots, s_n \rightarrow^? l_n\sigma, \sigma$$

where $l_i\sigma, 1 \leq i \leq n$, is bounded in depth by d (only those terms which contain variables become deeper after instantiation; however, when we substitute a variable, which could occur at at most below one function, by a term bounded in depth by $d-1$, we could get a term of depth at most d). Thus, each of the new subgoals is smaller (in $\succ$) than the original, and the proposition follows by induction on these smaller goals.

□