CONDITIONAL EQUATIONAL THEORIES AND COMPLETE SETS OF TRANSFORMATIONS*

Steffen HÖLLODBLER
FG Intellektik, FB Informatik, Technische Hochschule Darmstadt, Alexanderstrasse 10, 6100 Darmstadt, FRG

Abstract. The idea of combining the advantages of function and logic programming has attracted many researchers. Their work ranges from the integration of existing languages over higher-order logic to equational logic languages, where logic programs are augmented with equational theories. Recently, it has been proposed to handle those equational theories by complete sets of transformations. These transformations are extensions of the rules introduced by Herbrand and later used by Martelli and Montanari to compute the most general unifier of two expressions. We generalize this idea to complete sets of transformations for arbitrary conditional equational theories, the largest class of equational theories that admit a least Herbrand model. The completeness proof is based on the observation that each refutation with respect to linear paramodulation and reflection can be modelled by the transformations. As certain conditions imposed on an equational theory restrict the search space generated by paramodulation and reflection we can easily refine our transformations—due to the completeness proof—if the conditional equational theory is ground confluent or canonical.

1. Introduction

In recent years many proposals have been made to combine function and logic programming [4]. They range from the integration of existing languages, e.g. Loglisp [60], Qute [63], or LeFun [11], over higher-order logic (e.g. [WJ), to equational logic languages, where logic programs are augmented with equational theories, e.g. Eqlog [25].

In Loglisp, for example, a Horn clause calculus called Logic is implemented by a set of Lisp functions such that Logic and Lisp are mutually embedded. Loglisp is not a logically complete system but smoothly integrates the advantages of logic and functional programming by giving the user the possibility to submit Lisp data objects to a Horn clause theorem prover and vice versa.

If we view functional programming as deduction in a (typed) λ-calculus, then the natural way to combine functional and logic programming is to allow λ-terms as arguments of predicates. This idea goes back to Huet [36], who defines a complete calculus for higher-order logic. Recently, Miller and Nadathur [51] have recast this idea in a logic programming language called λ Prolog. To be complete, such a system

* This is a revised and extended version of a paper presented at the International Conference on Fifth Generation Computer Systems, 1988.
must incorporate a higher-order unification algorithm [38]. Unfortunately, higher-order unification is undecidable [37, 26] and many possible solutions have to be taken into account if two terms are to be unified whose initial symbols are function variables.

Besides an ongoing discussion of whether higher-order extensions of Prolog are needed [69], this last problem leads us to consider a first-order functional language. In such a language programs are defined by a set of first-order (conditional) equations (e.g. [54]). Following Robinson's idea [62] to remove troublesome (equational) axioms from the data base and to build them into the deductive machinery, Plotkin [57] has shown that these axioms can be handled by a sound and complete E-unification procedure. These results have been adapted for equational logic programming in [18, 19] (resp. [31]) in the case where a logic program is augmented by an unconditional (resp. conditional) equational theory.

Though the main semantic properties of logic programming such as the existence of a canonical domain of computation, the existence of a least and greatest model semantics, or the soundness and strong completeness for successful and finitely failed derivations of the underlying implementation model, hold also for equational logic programming (see [41, 42]), the main problem remains of how the E-unifiers of two expressions can be computed. This can be done by flattening and SLD-resolution (e.g. [3]), by paramodulation or special forms of it [61, 15, 58, 117], or by complete sets of transformations [45, 50, 20, 29]. Let us briefly recall these techniques.

Flattening a clause means to replace nested functional expressions by new variables and to add equations between the new variables and the replaced functional expressions to the clause. For example, an atom \( P(f(g(a)), b) \) can be flattened into

\[ P(x, b) \land x = f(y) \land y = g(z) \land z = a. \]

Cox and Pietrzykowski [10, 11] have proven that the flattening of a logic program together with the goal clause and the application of SLD-resolution subsumes the axioms of equality. Various modifications of this technique have been proposed. For example, Tamaki's [68] reducibility predicate is implemented using this technique. The reducibility predicate is nothing other than a directed equality predicate and to ensure the completeness of the system, the equational program in consideration must be confluent. Bosco et al. [8] have based a unification algorithm for canonical conditional theories on this technique. The principal disadvantage of flattening clauses is the lost possibility to reduce terms. Nutt et al. [53], for example, have demonstrated that reducing clauses may cut an infinite search space to a finite one. To overcome this problem in Leaf, for example, Barbuti et al. [3] imposed an annotation on variables, i.e. variables may be in input or output mode, and designed a complex computation rule and a new inference rule to simulate reduction.

Paramodulation (or special forms of it) is based on the idea that terms in a clause can be replaced by equal ones. Unfortunately, there are generally many terms to which paramodulation can be applied. Morris [52] and Anderson [2] have shown
that paramodulation needs to be applied only to selected literals until they become syntactically unifiable. Many efforts have been made to reduce the number of occurrences to which paramodulation has to be applied. For example, Hullot [39] has shown that it suffices to consider only so-called basic occurrences if the equational theory is a canonical term rewriting system. Furthermore, we may consider only innermost basic occurrences if in addition the term rewriting system is completely defined [15]. On the other hand, Echahed [133] has proven that for canonical term rewriting systems, whose left-hand sides are pairwise not strictly subunifiable, it suffices to apply narrowing only to a single term.

One disadvantage of paramodulation is that it may be applied to proper subterms of an expression without paying attention to the outer symbols. For example, to solve the problem of whether \( f(s) \) and \( g(t) \) are unifiable under a certain equational theory, where \( s \) and \( t \) are terms, it may be possible to apply paramodulation infinitely many times to subterms of \( s \) and \( t \) without solving the problem. On the other hand, the problem is only solvable if we can “replace” the initial function symbol \( f \) by \( g \) or vice versa. For example, if our equational theory contains the equation \( f(s') = g(t') \), then we may replace our initial problem by the two subproblems of whether \( s \) and \( s' \) (resp. \( t \) and \( t' \)) are unifiable. This has been the key observation which has led to the partial unification procedure for graph based equational reasoning developed in [7, 6]. It has also been the main motivation for the development of universal unification procedures based on complete sets of transformations.

These transformations are extensions of the rules introduced by Herbrand [27] and later used by Martelli and Montanari [49] to compute the most general unifier of two expressions. In their approach computing the most general unifier of the expressions \( s \) and \( t \) is equivalent to solving the set \( \{s \equiv t\} \) of equations. Such a set is solved by transforming it into an equivalent solved form. For example,

\[
\{f(x, x) \equiv f(a, y)\}
\]

is transformed into

\[
\{x \equiv a, x \equiv y\}.
\]

The variable \( x \) can now be replaced by the constant \( a \) in the second equation and we obtain

\[
\{x \equiv a, a \equiv y\}.
\]

Reversing the elements of the second equation yields the solved form

\[
\{x \equiv a, y \equiv a\}
\]

from which the most general unifier \( \{x \leftarrow a, y \leftarrow a\} \) for the initial expressions can be read off. In the presence of equational axioms like, for example, \( f(z, b) \equiv c \), \( \{x \leftarrow a, y \leftarrow a\} \) is still a solution for the unification problem but there are other independent unifiers like \( \{x \leftarrow b, y \leftarrow b\} \). These unifiers can be computed using additional transformations taking into account (variants of) the equational axioms.
of the theory in consideration. In our example,
\[ \{ f(x, x) \equiv f(a, y) \} \]
is also transformed into
\[ \{ x \equiv z, x \equiv h, f(a, y) \equiv c \} \]
and
\[ \{ x \equiv z, x \equiv b, a \equiv z', y \equiv b, c \equiv c \}. \]
By removing the trivial equation \( c \equiv c \), eliminating variables, and reversing the elements of equations as in the previous example we obtain the solved set
\[ \{ x \equiv b, y \equiv b, z \equiv b, z' \equiv a \} \]
from which the unifier can again be read off.

The additional transformations, however, are so far only defined for unconditional equational theories. These theories are not the largest class that admit a least Herbrand model and an initial semantics. This is the class of conditional or Horn equational theories [48]. In this paper we define complete sets of transformations for arbitrary conditional equational theories. To prove the completeness we make use of the completeness results known for (linear) paramodulation [17] or special forms of it [30] and show by a simple proof that each refutation with respect to paramodulation and reflection can be modelled by the transformations. This proof allows us to refine our transformations if the equational theory is ground confluent or canonical in much the same way as narrowing refines paramodulation. Finally, we show that for canonical theories rewriting can be applied as a simplification rule.

In the following section we briefly recall some basic notions and in Section 3 we give an account of the completeness results achieved for paramodulation and special forms of it. The transformations are introduced in Section 4 and the completeness proof is given in Section 5. In Section 6 we refine the transformation rules and we will finish by comparing our approach with others.

2. Preliminaries

We assume the reader to be familiar with logic programming (e.g. [47]), equations and rewrite rules (e.g. [35]), and unification theory [65, 66]. Throughout the paper we make use of the notational conventions laid down in Table 1 in the sense that, whenever we use \( x \), we implicitly assume that \( x \) is a variable. Set operators applied to multisets denote their multiset analogs. Furthermore, \( \text{Var}(X) \) denotes the set of variables occurring in the syntactic object \( X \).

<table>
<thead>
<tr>
<th>( a, b, \ldots )</th>
<th>constructors</th>
<th>( F )</th>
<th>multiset of equations</th>
<th>( x, y, \ldots )</th>
<th>variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>equation</td>
<td>( f, g, \ldots )</td>
<td>function symbols</td>
<td>( \sigma, \theta, \ldots )</td>
<td>substitutions</td>
</tr>
<tr>
<td>( EP )</td>
<td>equational program</td>
<td>( x, \ell, \ldots )</td>
<td>terms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1
An equation has the form \( \{s, t\} \) or \( \{t\} \). Expressions of the form \( \{s, t\} \) (resp. \( \{t\} \)) are interpreted as non-trivial (resp. trivial) equations \( s \equiv t \) (resp. \( t \equiv t \)). The labelled set notation has been introduced in [64] and emphasizes that the order in which the terms \( s \) and \( t \) are written in an equation is immaterial. For notational convenience we will commonly use the more usual form \( s \equiv t \) (resp. \( t \equiv t \)) to represent \( \{s, t\} \) (resp. \( \{t\} \)).

An equational program \( EP \) consists of a finite set of equational clauses of the form \( l \rightarrow r \iff F \). The arrow in the head of an equational clause emphasizes that equational clauses are used only from left-to-right. Let \( EP^{-1} = \{r \rightarrow l \iff F | l \rightarrow r \iff F \in EP\} \). A goal clause is a clause of the form \( \iff F \). To ease our notation we often omit the curly brackets in the body of an equational clause.

We only consider E-interpretation, i.e. interpretations which obey the axioms of equality as there are the axioms of reflexivity, symmetry, transitivity, and substitutivity (see e.g. [9]). The semantics of an equational program \( EP \) can be given as the least Herbrand model for \( EP \) together with the axioms of equality for \( EP \) [16]. By \( s =_{EP} t \) we denote that \( s \) and \( t \) are equal under \( EP \) or, equivalently, that \( s \equiv t \) is a logical consequence of \( EP \).

A substitution is defined to be a mapping from the set of variables into the set of terms which is equal to the identity almost everywhere. Hence, a substitution \( \sigma \) can be represented as the finite set of pairs \( \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\} \), where \( x_i \neq t_i, 1 \leq i \leq n \). \( \{x_i | 1 \leq i \leq n\} \) is called the domain and \( \{t_i | 1 \leq i \leq n\} \) is called the codomain of \( \sigma \). Substitutions are extended to morphisms on the set of terms and equations. \( \sigma|_V \) denotes the restriction of \( \sigma \) to the set \( V \) of variables. For notational convenience we assume that composition of substitutions precedes restriction, i.e. \( \sigma \theta|_V = (\sigma \theta)|_V \).

Two substitutions \( \sigma \) and \( \theta \) are said to be equal under \( EP \) with respect to the set \( V \) of variables, in symbols \( \sigma =_{EP} \theta[V] \) iff for all \( x \in V \) we find \( \sigma x =_{EP} \theta x \). \( \sigma \) is said to be more general modulo \( EP \) than \( \theta \) with respect to \( V \), in symbols \( \sigma \geq_{EP} \theta[V] \), iff there exists a substitution \( \lambda \) such that \( \lambda \sigma =_{EP} \theta[V] \). If the domain of \( \sigma \) is contained in \( V \), then we abbreviate \( \sigma \geq_{EP} \theta[V] \) to \( \sigma \geq_{EP} \theta \). \( \sigma \) and \( \theta \) are said to be variants iff \( \sigma \geq \theta \) and \( \theta \geq \sigma \).

An answer substitution for an equational program \( EP \) and a goal clause \( \iff F \) is a substitution for the variables occurring in \( F \). An answer substitution \( \sigma \) for \( EP \) and \( \iff F \) is said to be correct iff each element of \( \sigma F \) is a logical consequence of \( EP \).

Since we introduce several new inference rules we assume that derivations and refutations are defined with respect to a set of inference rules. If there exists a refutation of \( EP \cup \{\iff F\} \) with respect to some set of inference rules using substitutions \( \sigma_1, \ldots, \sigma_n \), then \( \sigma_n \ldots \sigma_1|_{\text{Var}(F)} \) is said to be a computed answer substitution.

3. Paramodulation

Paramodulation has been invented by Robinson and Wos [61] as a substitution rule for first order theories with equality. Furbach et al. [17] have recast (linear)
paramodulation as an inference rule for conditional equational theories. Let $G$ be
the goal clause $\leftarrow F \cup \{E\}$, $P = l \rightarrow r \leftarrow F^*$ be a new variant of a program clause,
$s$ be a subterm of $E$, and $E'$ be the equation obtained from $E$ by replacing the $s$
by $r$. If $s$ and $l$ are unifiable with most general unifier $\sigma$, then $G' = \leftarrow \sigma (F \cup \{E'\} \cup F^*)$
is called the paramodulant of $G$ and $P$, in symbols $G \rightarrow_{p(E,s,P,\sigma)} G'$. We say that
paramodulation is applied to an element of an equation $s \equiv t$ if either $s$ or $t$ has
been replaced.

To express syntactic equality we have to use the axiom of reflexivity. Let $G$ be
the goal clause $\leftarrow F \cup \{s \equiv t\}$. If $s$ and $t$ are unifiable with most general unifier $\sigma$, then $G' = \leftarrow \sigma F$ is called reflectant of $G$, in symbols, $G \rightarrow_{r(s \equiv t,\sigma)} G'$.

As the following example shows we need the functional reflexive axioms or,
equivalently, an instantiation rule to ensure the completeness of paramodulation. Let $G$ be
the goal clause $\leftarrow F \cup \{E\}$, $x$ be a variable in $E$, $f$ be an $n$-ary function
symbol, $x_1, \ldots, x_n$ be new variables, and $\sigma = \{x \leftarrow f(x_1, \ldots, x_n)\}$, then $G' = \sigma G$ is
called instance of $G$, in symbols, $G \rightarrow_{\{E,\sigma\}} G'$.

For notational convenience we often omit $E$, $s$, $P$, or $\sigma$ when writing derivations
if they can be determined by the context. Furthermore, we underline the selected
subgoal or subterm.

As an example consider the equational program

\[
\text{FUN}: g \rightarrow a \quad (g)
\]
\[
f(c(g), c(a)) \rightarrow d(c(g), c(a)) \leftarrow (f)
\]

and the question of whether there exists a substitution $\theta$ such that $\theta f(x, x) \equiv \theta d(x, x)$
is a logical consequence of FUN. This question can be answered with $\theta = \{x \leftarrow c(g)\}$
by the refutation in Fig. 1.

\[
\leftarrow \theta f(x, x) \equiv d(x, x) \rightarrow_{\{x \leftarrow c(y)\}} \leftarrow \theta f(c(y), c(y)) \equiv d(c(y), c(y))
\]
\[
\rightarrow_{p(f)} \leftarrow \theta d(c(g), c(a)) \equiv d(c(g), c(a))
\]
\[
\rightarrow_{p(g)} \leftarrow d(c(g), c(a)) \equiv d(c(g), c(a))
\]
\[
\rightarrow_{r} \square.
\]

Fig. 1.

The interested reader may verify that without the instantiation rule a refutation
of $\leftarrow f(x, x) \equiv d(x, x)$ is impossible. Formally, the need for the instantiation rule
comes from the lifting lemma which states that, if there exists a refutation of
$E \cup \{\leftarrow \sigma F\}$ with computed answer substitution $\theta$, then there exists a refutation of
$E \cup \{\leftarrow F\}$ and furthermore, if $\gamma$ is the computed answer substitution of this
refutation, then $\gamma$ is more general than $\theta \sigma$. In the proof of the lifting lemma, one
is confronted with the case that in the refutation of $EP \cup \{\leftarrow \sigma F\}$ paramodulation is applied to a term $s$ which was introduced by $\sigma$. To be able to apply the respective paramodulation step to $\leftarrow F$ we have to instantiate $\leftarrow F$. As an example consider the paramodulation step

$$\leftarrow_1[f(x, x) \leftarrow d(x, x)] = \leftarrow_1[f(c(g), c(g)) \leftarrow d(c(g), c(g))]$$

which was lifted in Fig. 1 using an instantiation and a paramodulation step.\(^1\)

This use of the instantiation rule suggests the definition of a new inference rule, instantiation and paramodulation ($\rightarrow_{\text{ip}}$): $G \rightarrow_{\text{ip}(\sigma)} G'$ iff $G'$ has been obtained from $G$ by a (possibly empty) finite sequence of instantiation steps followed by a single paramodulation step and, if $\sigma_1, \ldots, \sigma_n$ are the substitutions used in this derivation, then $\sigma = \sigma_n \ldots \sigma_1$. For example, the first two inference steps in Fig. 1 can be comprised to

$$\leftarrow_1[f(x, x) \leftarrow d(x, x)] \rightarrow_{\text{ip}((x \leftarrow c(g)) \leftarrow d(c(g), c(g)))} \leftarrow_1[f(c(g), c(a)) \leftarrow d(c(g), c(g)))].$$

Note, an instantiation and paramodulation step corresponds to a paramodulation step using a prefixed axiom in [55].

The completeness of reflection, instantiation and paramodulation follows immediately from [17].

**Theorem 3.1** (Completeness of $\{\rightarrow_{\text{ip}}, \rightarrow_{\text{r}}\}$). If $\theta$ is a correct answer substitution for $EP$ and $\leftarrow F$, then there exists a computed answer substitution $\sigma$ obtained by a refutation of $EP \cup EP^{-1} \cup \{\leftarrow F\}$ with respect to $\{\rightarrow_{\text{ip}}, \rightarrow_{\text{r}}\}$ such that $\sigma \models \theta$.

Clauses from $EP^{-1}$ are needed in Theorem 3.1, since we cannot generally assume that arbitrary equational programs are ground confluent (see FUN). In analogy to the respective result for SLD-resolution (e.g. [47]) it can easily be proven that refutations with respect to reflection, instantiation and paramodulation are independent of a computation rule, i.e. a function which applied to a non-empty goal clause always selects an equation from that clause. Recall, two substitutions $\sigma$ and $\theta$ are variants iff $\sigma$ is more general than $\theta$ and vice versa.

**Theorem 3.2** (Independence of the computation rule, Hölldobler [32]). Let $R$ and $R'$ be computation rules. If there exists a refutation of $EP \cup \{\leftarrow F\}$ with respect to $\{\rightarrow_{\text{ip}}, \rightarrow_{\text{r}}\}$, computed answer substitution $\sigma$, and via $R$, then there exists a refutation of $EP \cup \{\leftarrow F\}$ with respect to $\{\rightarrow_{\text{ip}}, \rightarrow_{\text{r}}\}$, via $R'$, and, if $\sigma'$ is the computed answer substitution of the refutation via $R'$, then $\sigma$ and $\sigma'$ are variants. Furthermore, in both refutations paramodulation is applied the same number of times.

\(^1\) One should observe that the instantiation rule is necessary since we apply paramodulation linearly, i.e. only to goal clauses. If we define a ground linear simplification ordering and allow the application of paramodulation also to program clauses, then the instantiation rule is obsolete and, moreover, paramodulation need only be applied to non-variable terms (see [56, 341]).
Remark 3.3. If the computation rule selects a subgoal of the form \( x \rightleftharpoons y \), where \( x \) and \( y \) are variables, then it does not suffice to apply only reflection but we have also to apply instantiation and paramodulation. However, due to Theorem 3.2, we may choose a computation rule that never selects an equation of the form \( x \rightleftharpoons y \) if it has another choice. In such a refutation we will eventually encounter a goal clause of the form \( \left\{ x_i \rightleftharpoons y_i \mid 1 \leq i \leq n \} \) and it is easy to see that the completeness of reflection, instantiation and paramodulation is retained even if we apply only reflection to solve such a goal clause. However, in this case we find for each correct answer substitution \( \theta \) for \( EP \) and \( \left\{ F \} \) a computed answer substitution \( \sigma \) which is more general modulo \( EP \) than \( \theta \) (see [32]).

Of course, the search space generated by reflection, instantiation and paramodulation contains far too many redundant and irrelevant inferences and it has been proposed in [67, 46] to impose certain conditions on equational theories such that paramodulation need not be applied to variable occurrences. This restricted form of paramodulation is often called narrowing (e.g. [39]). Obviously, instantiation is no longer needed if it suffices to apply paramodulation to non-variable terms.

In [30] these refinements of paramodulation have formally been developed for conditional equational theories. It has been shown that clauses from \( EP^{-1} \) are no longer needed if the equational program is ground confluent. \( EP \) is said to be ground confluent iff for all ground goal clauses \( G, G_1, G_2 \) such that \( G_1 \leftarrow G \rightarrow G_2 \) there is a goal clause \( G' \) such that \( G_1 \leftarrow G' \leftarrow G_2 \), where \( \leftarrow \) denotes a derivation with respect to reflection, instantiation and paramodulation. Furthermore, (conditional) narrowing \( (\rightarrow_n) \) can be applied instead of instantiation and paramodulation if the equational program is a non-trivial and ground confluent term rewriting system and the answer substitution is in normal form. An equational program is said to be non-trivial iff it does not contain a trivial clause, i.e. a clause of the form \( x \rightleftharpoons r \leftarrow F \).

A term rewriting system is an equational program, where for each clause \( l \rightarrow r \leftarrow F \) we find that each variable occurring in \( F \) and \( r \) occurs also in \( l \). A substitution \( \theta \) is said to be in normal form with respect to a term rewriting system \( EP \) iff there does not exist a term \( t \) in the codomain of \( \theta \), a rewrite rule \( l \rightarrow r \leftarrow F \) in \( EP \), and a substitution \( \sigma \) such that \( t = \sigma l \) and each ground instance of \( \theta F \) is a logical consequence of \( EP \).

To exemplify these definitions consider the following simple equational program

\[
CREDiT: \quad \text{paid (mary)} \rightarrow \text{yes} \leftarrow (p1) \\
\text{paid (john)} \rightarrow \text{no} \leftarrow (p2) \\
\text{credibility (x)} \rightarrow \text{high} \leftarrow \text{paid (x)} = \text{yes} (c1) \\
\text{credibility (x)} \rightarrow \text{low} \leftarrow \text{paid (x)} = \text{no} (c2).
\]

This program states that the credibility of a customer is high if she or he has paid his last bills immediately and it is low otherwise. Mary has paid her bills immediately,
whereas John, for reasons we are not aware of, has not paid some of his recent bills. Obviously, this program is a non-trivial term rewriting system. The substitution \( \theta = \{ y \leftarrow \text{credibility}(\text{john}) \} \) is not in normal form with respect to \( \text{CREDIT} \) since \( \text{CREDIT} \) entails \( \text{paid}(\text{john}) = \text{no} \) and with \( \sigma = \{ x \leftarrow \text{john} \} \) we find that the left-hand side of the head of (c2) matches \( \text{credibility}(\text{john}) \). However, \( \theta' = \{ y \leftarrow \text{low} \} \) is in normal form with respect to \( \text{CREDIT} \) and is called the normal form of \( \theta \) with respect to \( \text{CREDIT} \).

The set of function symbols is divided by a term rewriting system into two disjoint subsets, the set of defined function symbols and the set of constructors. \( f \) is said to be a defined function symbol iff the term rewriting system contains a rule for \( f \).

In our \( \text{CREDIT} \) example we have the defined function symbols \( \text{paid} \) and \( \text{credibility} \) whereas \( \text{mary}, \text{john}, \text{yes}, \text{no}, \text{high} \) and \( \text{low} \) are constructors. Since these constructors are nullary they are also called constants.

The following theorems are immediate consequences of [30] and Theorem 3.2.

**Theorem 3.4.** Let \( \text{EP} \) be a ground confluent equational program and \( R \) be a computation rule. If \( \theta \) is a correct answer substitution for \( \text{EP} \) and \( \equiv_F \), then there exists an \( R \)-computed answer substitution \( \sigma \) obtained by a refutation of \( \text{EP} \cup \{ \equiv_F \} \) with respect to \( \{ \rightarrow_{ip}, \rightarrow_r \} \) such that \( \sigma \supseteq \theta \).

**Theorem 3.5 (Strong completeness of \( \{ \rightarrow_n, \rightarrow_r \} \)).** Let \( \text{EP} \) be a ground confluent and non-trivial term rewriting system and \( R \) be a computation rule. If \( \theta \) is a normalized correct answer substitution for \( \text{EP} \) and \( \equiv_F \), then there exists an \( R \)-computed answer substitution \( \sigma \) obtained by a refutation of \( \text{EP} \cup \{ \equiv_F \} \) with respect to \( \{ \rightarrow_n, \rightarrow_r \} \) such that \( \sigma \supseteq \theta \).

As a consequence, narrowing and reflection is complete for canonical conditional term rewriting systems and rewriting can be applied as a simplification rule, where a goal clause \( G \) rewrites to \( G' \) iff \( G \rightarrow_{\rho(\sigma)} G' \) and \( \sigma \) does not bind a variable in \( G \). Note, this definition differs from the one given in e.g. [5] or [43]. The reason is that we are mainly interested in equation solving and the conditions of a rewrite rule applied are simply added to the new goal clause. Recalling the \( \text{CREDIT} \) example we find that

\[
\equiv \text{credibility}(y) \equiv \text{high}
\]

can be rewritten to

\[
\equiv \text{high} \equiv \text{high} \land \text{paid}(y) \equiv \text{yes}
\]

using (c1) and substitution \( \{ x \leftarrow y \} \). After eliminating the trivial equation \( \text{high} \equiv \text{high} \) and applying narrowing to \( \text{paid}(y) \) using (p1) and the substitution \( \{ y \leftarrow \text{mary} \} \) we obtain the goal clause

\[
\equiv \text{yes} \equiv \text{yes}.
\]
This goal can be solved by applying reflection and we obtain the answer that Mary's credibility is high.

One should observe that the question of whether a goal clause can be rewritten is decidable, whereas the same question is undecidable if we consider, for example, Kaplan's rewrite relation (see [43]). Moreover, we may apply other simplification rules such as removal of trivial equations, decomposition of decomposable equations [45], and elimination of variables if the goal clause contains an equation of the form \( x \equiv t \) and no defined function symbol occurs in \( t \).

Despite these refinements, linear paramodulation bears several disadvantages. It is only complete if we add the functional reflexive axioms to a program or use an additional instantiation rule. There are generally several terms occurring in a literal to which paramodulation or its special forms are applicable and we have to investigate all of them to ensure completeness. These difficulties can be overcome if we use transformation rules.

4. The transformations

As we have mentioned in the Introduction the transformation rules are an extension of the rules invented by Herbrand [27] and Martelli and Montanari [49] to compute the most general unifier of two expressions. Therefore, we briefly repeat these rules. However, in contrast to the Introduction we define these transformations as inference rules and the unifier of an initial set of equations is obtained as computed answer substitution.

The term decomposition (\( \rightarrow_d \)) rule decomposes an equation of the form \( f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n) \) into the set of corresponding arguments, i.e.

\[
\iff F \cup \{ f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n) \} \rightarrow_d \iff F \cup \{ s_i \equiv t_i | 1 \leq i \leq n \}.
\]

The variable elimination (\( \rightarrow_v \)) rule applied to a goal clause \( \iff F \cup \{ x \equiv t \} \) eliminates the variable \( x \) by replacing each occurrence of \( x \) by \( t \) if \( x \) does not occur in \( t \), i.e.

\[
\iff F \cup \{ x \equiv t \} \rightarrow_v \iff F \cup \{ x \leftarrow t \}.
\]

The rule removal of trivial equations (\( \rightarrow_t \)) removes a trivial equation, i.e.

\[
\iff F \cup \{ t \equiv t \} \rightarrow_t \iff F.
\]

A reflection step can be modelled by a sequence of \( \rightarrow_t, \rightarrow_v, \) and \( \rightarrow_d \) steps [49]. These transformations achieve syntactic unification, whereas the following three rules are only applicable with respect to an equational program.

The lazy narrowing (\( \rightarrow_m \)) rule applied to an equation of the form \( f(s_1, \ldots, s_n) \equiv s_{n+1} \) and using an equational clause \( f(t_1, \ldots, t_n) \rightarrow t_{n+1} \iff F^* \) forces the comparison of corresponding arguments and right-hand sides, i.e.

\[
\iff F \cup \{ f(s_1, \ldots, s_n) \equiv s_{n+1} \} \rightarrow_m \iff F \cup F^* \cup \{ s_i \equiv t_i | 1 \leq i \leq n + 1 \}.
\]
We have called this rule lazy, since the corresponding arguments are not immediately solved but added to the new goal clause and, hence, can be handled according to the overall strategy encoded in a computation rule.

The rule \textit{paramodulation upon variables} ($\rightarrow_{pv}$) applied to an equation of the form $x \doteq s$ and using an equational clause $f(t_1, \ldots, t_n) \rightarrow r \equiv F^*$ instantiates $x$ to $f(x_1, \ldots, x_n)$ and, then, forces the comparison of corresponding arguments and right-hand sides, i.e.

$$\leftrightarrow F \cup \{x \doteq s\} \rightarrow_{pv(\sigma)} \leftrightarrow F \cup F^* \cup \{x_i \doteq \sigma t_i \mid 1 \leq i \leq n\} \cup \{s \doteq r\},$$

where $\sigma = \{x \leftarrow f(x_1, \ldots, x_n)\}$, $x_1, \ldots, x_n$ are new variables, and $s$ is a non-variable term.

So far we cannot use trivial clauses. The rule \textit{application of a trivial clause} ($\rightarrow_{tc}$) applied to an equation $s \doteq t$ and using a trivial clause $x \rightarrow r \equiv F^*$ forces the comparison of corresponding left- (resp. right-) hand sides, i.e.

$$\leftrightarrow F \cup \{s \doteq t\} \rightarrow_{tc} \leftrightarrow F \cup F^* \cup \{s \doteq x, r \doteq t\}.$$

As we will learn from the proof of the completeness of the transformation rules, neither $\rightarrow_{in}$, nor $\rightarrow_{pv}$, nor $\rightarrow_{tc}$ need to be applied to $s \doteq x$ anymore.

Due to the lazy nature of the transformation rules introduced so far, lazy narrowing can only be applied to the elements of an equation but not to proper subterms of these elements. This would lead to an incompleteness of our transformation rules. Suppose the only program clause is $f(x) \rightarrow a \leftarrow$ and consider the goal clause $\leftarrow y \doteq c(f(y))$. In a refutation with respect to narrowing and reflection, narrowing can be applied to $f(y)$ yielding $\leftarrow y \doteq c(a)$ which can be solved by binding $c(a)$ to $y$. However, this refutation cannot be modelled by the transformation rules introduced so far.

The \textit{imitation} ($\rightarrow_{im}$) rule applied to an equation of the form $x \doteq f(t_1, \ldots, t_n)$ instantiates $x$ to $f(x_1, \ldots, x_n)$ and, then, forces the comparison of corresponding arguments, i.e.

$$\leftrightarrow F \cup \{x \doteq f(t_1, \ldots, t_n)\} \rightarrow_{im(\sigma)} \leftrightarrow F \cup \{x_i \doteq \sigma t_i \mid 1 \leq i \leq n\},$$

where $\sigma = \{x \leftarrow f(x_1, \ldots, x_n)\}$ and $x_1, \ldots, x_n$ are new variables. In our example,

\[
\begin{align*}
\leftarrow y \doteq c(f(y)) & \rightarrow_{im(y \leftarrow c(z))} \leftarrow z \doteq f(c(z)) \\
& \rightarrow_{in} \leftarrow z \doteq a, c(z) \doteq x \\
& \rightarrow_{v(z \leftarrow a)} \leftarrow c(a) \doteq x \\
& \rightarrow_{v} \square.
\end{align*}
\]

\textbf{Notation 4.1.} In the sequel let \textit{TRANS} = \{$\rightarrow_{d}, \rightarrow_{v}, \rightarrow_{1}, \rightarrow_{in}, \rightarrow_{pv}, \rightarrow_{tc}, \rightarrow_{im}$\}.

The transformation rules can be divided into three classes. The \textit{unification rules} perform term decomposition, variable elimination, and removal of trivial equations, the \textit{lazy
paramodulation rules lazy narrowing, paramodulation upon variables, and application of a trivial clause, and the imitation rule. It should be noted that we have no transformation which corresponds to the instantiation rule in the sense that an uninformed choice for the binding of a variable has to be made. The rules paramodulation upon variables and imitation instantiate a variable $x$ by $f(x_1, \ldots, x_n)$, but whenever such a rule is applied we know that the binding for $x$ has to be of the form $f(t_1, \ldots, t_n)$.

Our transformations can be regarded as an extension of the rules BT given in [21] to conditional equational theories. They differ if the selected equation is of the form $x = t$. If $t$ is a variable, then Gallier and Snyder provide an additional transformation, which instantiates the goal clause by $\{x \leftarrow f(x_1, \ldots, x_n)\}$. If $t$ is not a variable, then Gallier and Snyder apply lazy paramodulation rules only to $t$. As an example consider the term rewriting system $\text{INF} = \{f \rightarrow c(f) \iff \}$. Then,

$$\iff x = c(x) \rightarrow \_in \iff x = f, x = f \rightarrow c(x \leftarrow f), \iff f \rightarrow \_in, \square,$$

and in the lazy narrowing step the equational clause $c(f) \rightarrow f \iff \in \text{INF}^{-1}$ is used. This is the only way Gallier and Snyder can solve the initial goal clause since paramodulation upon variables cannot be applied due to the chosen restriction, variable elimination cannot be applied since the occur check fails, and an imitation yields a variant of $x = c(x)$. The strange observation about Gallier and Snyder's restricted use of lazy paramodulation rules is that even if the equational program is ground confluent (like INF) they have to use equational clauses in both directions, whereas without the restriction we find

$$\iff x = c(x) \rightarrow \_in \iff c(f) = c(f) \rightarrow \_in, \square.$$

It is easy to see that the transformations are sound. Each derivation step with respect to TRANS can be modelled by a sequence of resolution steps using the axioms of equality. For example, a lazy narrowing step can be modelled by four resolution steps using the axiom of transitivity twice, the respective equational clause, and a substitutivity axiom as the following example shows. Let $f(x') \rightarrow a \iff$ be the equational clause used in the lazy narrowing step

$$\iff f(c(z)) = z \rightarrow \_in \iff c(z) = x', z = a.$$  

Then, we obtain

$$\iff f(c(z)) = z \rightarrow \iff f(c(z)) = x, x = z$$

$$\rightarrow \iff f(c(z)) = x, x = y, y = z$$

$$\rightarrow \iff f(c(z)) = f(x'), a = z$$

$$\rightarrow \iff c(z) = x', a = z,$$

where $\rightarrow$ denotes a resolution step and $x, y$ are new variables introduced by the
Theorem 3.2.

The following technical propositions show how some refutations with respect to reflection, instantiation and paramodulation can be transformed into simpler ones with respect to their complexity.
**Proposition 5.1.** Let \( E = f(s_1, \ldots, s_m) \models f(t_1, \ldots, t_n) \). If there exists a refutation of \( EP \cup \{ \leftarrow F \cup \{ E \} \} \) with respect to \( \{ \rightarrow_{ip}, \rightarrow_r \} \), computed answer substitution \( \theta \) and complexity \( M = \langle \#p, D(\theta), \#s, \#e \rangle \), where paramodulation is never applied to an element of a descendant of \( E \), then there exists a refutation of \( EP \cup \{ s_i \models t_i \mid 1 \leq i \leq n \} \) with respect to \( \{ \rightarrow_{ip}, \rightarrow_r \} \), computed answer substitution \( \theta \) and complexity \( \langle \#p, D(\theta), \#s-2, \#e + m-1 \rangle \leq M \).

**Proof.** Without loss of generality we may assume that \( E \) is the first selected equation in the refutation. The result is proven by induction on the number \( n \) of instantiation and paramodulation steps applied to the descendents of \( E \). The case \( n = 0 \) being trivial we turn to the induction step and assume that the result holds for \( n \). Suppose, instantiation and paramodulation is applied \( n+1 \) times to descendents of \( E' = f(s'_1, \ldots, s'_m) \models f(t'_1, \ldots, t'_n) \) in a refutation of \( \leftarrow F' \cup \{ E' \} \) with respect to reflection, instantiation and paramodulation, computed answer substitution \( \theta' \), and complexity \( M' \). Since paramodulation is never applied to an element of a descendant of \( E' \), we may assume that the first instantiation and paramodulation step is applied to a proper subterm of an element of \( E' \), say \( s'_j \), transforming it into \( s''_j \). Let \( \sigma \) be the substitution and \( l \to r \leftarrow F^* \) be the equational clause used in this step, \( F = \sigma(F' \cup F^*) \), \( E = f(s_1, \ldots, s_n) \models f(t_1, \ldots, t_n) = f(\sigma s'_1, \ldots, s''_n, \ldots, \sigma s'_n) \models f(\sigma t'_1, \ldots, \sigma t'_n) \). Then

\[
\leftarrow F' \cup \{ E' \} \to ip \leftarrow \sigma(F' \cup F^*) \\
\cup \{ f(\sigma s'_1, \ldots, s''_n, \ldots, \sigma s'_n) \models f(\sigma t'_1, \ldots, \sigma t'_n) \}
\]

and there exists a refutation of \( \leftarrow F \cup \{ E \} \) with respect to \( \{ \rightarrow_r, \rightarrow_{ip} \} \), with computed answer substitution \( \theta \) and complexity \( M \). From the induction hypothesis we learn that there exists a refutation of \( \leftarrow F \cup \{ s_i \models t_i \mid 1 \leq i \leq n \} \) with respect to \( \{ \rightarrow_r, \rightarrow_{ip} \} \), computed answer substitution \( \theta \) and complexity \( N < M \). Obviously,

\[
\leftarrow F' \cup \{ s'_i \models t'_i \mid 1 \leq i \leq n \} \to ip \leftarrow F \cup \{ s_i \models t_i \mid 1 \leq i \leq n \}
\]

and the result follows immediately. \( \Box \)

As an example consider the rewrite rule \( f(a) \to b \leftarrow \) and the refutation

\[
\leftarrow c(f(x)) \models c(b) \to ip(x \leftarrow a) \leftarrow c(b) \models c(b) \to r \square
\]

with computed answer substitution \( \{ x \leftarrow a \} \) and complexity \( \langle 1, \{ 1 \}, 5, 1 \rangle \). Then we find a refutation

\[
\leftarrow f(x) \models b \to ip(x \leftarrow a) \leftarrow b \models b \to r \square
\]

with computed answer substitution \( \{ x \leftarrow a \} \) and complexity \( \langle 1, \{ 1 \}, 3, 1 \rangle \leq \langle 1, \{ 1 \}, 5, 1 \rangle \).
The second technical proposition shows how the first paramodulation step applied to an element of a descendant of an equation in a refutation with respect to reflection, instantiation and paramodulation can be simulated.

**Proposition 5.2.** Suppose there exists a refutation of $EP \cup \{ \leftarrow F \cup \{s \rightleftharpoons t\} \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_t \}$, computed answer substitution $\theta$, and complexity $M = (\#p, D(\theta), \#s, \#e)$, where paramodulation is applied to an element, say $s'$, of a descendant $s \rightleftharpoons t'$ of $s \rightleftharpoons t$. Let $l \rightarrow r \leftarrow F^*$ be the program clause used in the first of these applications. Then there exists a refutation of $EP \cup \{ \leftarrow F \cup F^* \cup \{s \rightleftharpoons l, r \rightleftharpoons t\} \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_t \}$, computed answer substitution $\theta'$, and complexity $(\#p - 1, D(\theta'), \#s', \#e') \ll M$ such that $\theta'|_{\text{Var}(F \cup \{s \rightleftharpoons t\})} = \theta$.

The proof of this proposition is a variation of the proof of Proposition 5.1. The interested reader may verify that in the refutation of $EP \cup \{ \leftarrow F \cup F^* \cup \{s \rightleftharpoons l, r \rightleftharpoons t\} \}$ paramodulation need not be applied to an element of a descendant of $s \rightleftharpoons t$. It follows that Propositions 5.1 and 5.2 can be combined such that under the conditions of Proposition 5.2 a lazy narrowing step can be applied and we find a refutation of the newly generated goal clause with respect to reflection, instantiation and paramodulation having a complexity smaller than $M$.

As an example consider again the refutation in Fig. 1. This refutation yields the computed answer substitution $\theta = \{x \leftarrow c(g)\}$ and has complexity $M = (3, [2], 6, 1)$. Since in the third step of this refutation paramodulation is applied to an element, $f(c(g), c(a))$, of a descendant of $f(x, x) \rightleftharpoons d(x, x)$ for the first time, we find by Proposition 5.2 a refutation of $\leftarrow f(x, x) \rightleftharpoons f(c(g), c(a)), d(c(g), c(a)) \rightleftharpoons d(x, x)$ with respect to reflection, instantiation and paramodulation:

\[
\begin{align*}
\leftarrow f(x, x) &= f(c(g), c(a)), d(c(g), c(a)) \rightleftharpoons d(x, x) \\
\rightarrow_{(p(g, \{x \leftarrow c(g)\}) \leftarrow )} f(c(g), c(a)) &= f(c(g), c(a)), \\
&d(c(g), c(a)) \rightleftharpoons d(c(g), c(g)) \\
\rightarrow_r f(c(g), c(a)) &= d(c(g), c(g)), \\
\rightarrow_{p(g)} d(c(g), c(a)) &= d(c(g), c(g)) \\
\rightarrow_r d(c(g), c(a)) &= d(c(g), c(g)) \\
\rightarrow_{p(g)} d(c(g), c(a)) &= d(c(g), c(g)) \\
\rightarrow_r d(c(g), c(a)) &= d(c(g), c(g)) \\
\rightarrow \Box.
\end{align*}
\]

This refutation yields computed answer substitution $\theta' = \theta$ and has complexity $M' = (2, [2], 16, 2) \ll M$. Since paramodulation is never applied to an element of a descendant of $f(x, x) \rightleftharpoons f(c(g), c(a))$ we find by Proposition 5.1 the following refutation with respect to reflection, instantiation and paramodulation:

\[
\begin{align*}
\leftarrow x \rightleftharpoons c(g), x \rightleftharpoons c(a), d(c(g), c(a)) \rightleftharpoons d(x, x) \\
\rightarrow_{(r(\{x \leftarrow c(g)\}) \leftarrow )} c(g) &= c(a), d(c(g), c(a)) \rightleftharpoons d(c(g), c(g)) \\
\rightarrow_{p(g)} c(a) &= c(a), d(c(g), c(a)) \rightleftharpoons d(c(g), c(g)) \\
\rightarrow_r d(c(g), c(a)) &= d(c(g), c(g)) \\
\rightarrow_{p(g)} d(c(g), c(a)) &= d(c(g), c(a)) \\
\rightarrow_r d(c(g), c(a)) &= d(c(g), c(g)) \\
\rightarrow \Box.
\end{align*}
\]
This refutation yields computed answer substitution $\theta$ and has complexity $(2, \{2\}, 14, 3) \ll M' \ll M$. Observe, lazy narrowing applied to $\leftarrow f(x, x) \not= d(x, x)$ and using $f(c(g), c(a)) \leftarrow d(c(g), c(a)) \leftarrow$ yields precisely the initial goal clause of the previous refutation.

The last technical proposition is used to show that after an application of imitation or paramodulation upon variables we find a refutation of the new goal clause with respect to reflection, instantiation and paramodulation which yields a more general answer substitution and has a smaller complexity.

**Proposition 5.3.** Suppose there exists a refutation of $EP \cup \{ \leftarrow F \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_{r} \}$, computed answer substitution $\theta$ and complexity $M$. Suppose $x \leftarrow f(t_1, \ldots, t_n) \in \theta$ and let $x_i, 1 \leq i \leq n$, be new variables and $\gamma = \{ x \leftarrow f(x_1, \ldots, x_n) \}$. Then there exists a refutation of $EP \cup \{ \leftarrow_{\gamma} F \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_{r} \}$, computed answer substitution $\theta'$, and complexity $M'$ such that

$$\theta' \gamma|_{\text{Var}(F)} \gg \theta$$

and $M' \ll M$.

**Proof.** If there exists a refutation of $EP \cup \{ \leftarrow F \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_{r} \}$ and computed answer substitution $\theta$, then there exists a refutation of $EP \cup \{ \leftarrow_{\theta} F \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_{r} \}$ and empty computed answer substitution. In both refutations the number of applications of paramodulation, $\# p$, is the same. By the lifting lemma for reflection, instantiation and paramodulation [32] we find a refutation of $EP \cup \{ \leftarrow_{\gamma} F \}$ with respect to $\{ \rightarrow_{ip}, \rightarrow_{r} \}$ and computed answer substitution $\theta'$ such that $\theta' \gamma|_{\text{Var}(F)} \gg \theta$ and paramodulation is applied $\# p$ times. Since $D(\theta') \ll D(\theta)$ the result follows immediately. □

As an example consider the program clause $f(x) \rightarrow a \leftarrow$ and the refutation

$$\leftarrow y \not= c(f(y)) \rightarrow_{p(x \rightarrow y)} \leftarrow y \not= c(a) \rightarrow_{v(y \rightarrow c(a))} \Box.$$  

$\theta = \{ x \leftarrow c(a) \}$ is the computed answer substitution and $M = (1, \{2\}, 4, 1)$ the complexity of this refutation. Now, let $\gamma = \{ y \leftarrow c(z) \}$. Then

$$\leftarrow y \not= c(f(y))$$

$$= \leftarrow c(z) \not= c(f(c(z))) \rightarrow_{p(x \rightarrow c(z))} \leftarrow c(z) \not= c(a) \rightarrow_{r(z \rightarrow a)} \Box$$

with computed answer substitution $\theta' = \{ z \leftarrow a \}$ and complexity $(1, \{1\}, 6, 1) \ll M$. From Proposition 5.1 we learn that there exists a refutation of $\leftarrow z \not= f(c(z))$ with respect to reflection, instantiation and paramodulation, with the same computed answer substitution, but with smaller complexity. Observe, imitation applied to $\leftarrow y \not= c(f(y))$ yields $\leftarrow z \not= f(c(z))$.

It is easy to see that, if in the refutation of $EP \cup \{ \leftarrow F \}$ paramodulation is never applied to an element of a descendant of $E \in F$, then paramodulation is never applied to an element of a descendant of $\gamma E \in \gamma F$ in the refutation of $EP \cup \{ \leftarrow F \}$. 


We can now prove that for each refutation with respect to paramodulation, instantiation, and reflection there exists a corresponding refutation with respect to \(\text{TRANS}\) yielding a more general answer substitution. Recall, in a refutation with respect to reflection, instantiation and paramodulation we may apply a computation rule which never selects an equation of the form \(x \equiv y\) if it has another choice. Furthermore, if the goal clause contains only equations of this form then it suffices to apply reflection.

**Notation 5.4.** Let \(R^+\) be a computation rule that obeys this strategy.

**Theorem 5.5.** If there exists a refutation of \(EP \cup \{\leftarrow F\}\) with respect to \(\{\rightarrow \pi_p, \rightarrow r\}\) and computed answer substitution \(\theta\), then there exists a refutation of \(EP \cup \{\leftarrow F\}\) with respect to \(\text{TRANS}\) and via \(R^+\). Furthermore, if \(\sigma\) is the computed answer substitution of the refutation with respect to \(\text{TRANS}\), then \(\sigma \geq_{EP} \theta\).

**Proof.** The proof is by transfinite induction on the complexity \(M\) of the refutation of \(EP \cup \{\leftarrow F\}\) with respect to \(\{\rightarrow \pi_p, \rightarrow r\}\). We assume that the result holds for all \(M < M'\). Suppose

\[
\leftarrow F' \cup \{s \equiv t\} \rightarrow \square
\]

with respect to \(\{\rightarrow \pi_p, \rightarrow r\}\), computed answer substitution \(\theta'\), and complexity \(M' = (\#p', D(\theta'), \#s', \#e').\) Let \(s \equiv t\) be the first selected equation by \(R^+\). By Theorem 3.2 we may assume that \(s \equiv t\) is the first selected equation in (1).

1. If \(s\) and \(t\) are variables, then \(F'\) contains only equations of the form \(x \equiv y\) and we may assume that in (1) only reflection is applied. Reflection can be modelled by the rules \(\rightarrow \pi\), \(\rightarrow v\), and \(\rightarrow d\). Each application of one of these rules decreases the complexity of the refutation. Hence, the theorem follows immediately.

In the remaining cases we may assume that \(s\) or \(t\) is a non-variable term.

2. Suppose that in (1) paramodulation is applied to an element, say \(s'\), of a descendant \(s' \equiv t'\) of \(s \equiv t\). Let \(P = l \rightarrow r \leftarrow F^*\) be the program clause used in the first such application. By Proposition 5.2 we find

\[
\leftarrow F'' \cup \{s \equiv l, r \equiv t\} \cup F^* \rightarrow \square
\]

with respect to \(\{\rightarrow \pi_p, \rightarrow r\}\), computed answer substitution \(\theta^*\), and complexity \(M^*\) such that \(\theta^*|_{\text{Var}(F'' \cup \{s \equiv t\})} = \theta'\) and \(M^* \leq M'\). Recall, in (2) paramodulation need not be applied to an element of a descendant of \(s \equiv l\).

2.1 Suppose \(P\) is a trivial clause. Let \(F = F' \cup F^* \cup \{s \equiv l, r \equiv t\}\). Then,

\[
\leftarrow F'' \cup \{s \equiv t\} \rightarrow tc \leftarrow F
\]

and (2) ensures that there exists a refutation of \(EP \cup \{\leftarrow F\}\) with respect to \(\{\rightarrow \pi_p, \rightarrow r\}\).
computed answer substitution \( \theta = \theta^* \), and complexity \( M = M^* \ll M' \). The result follows by the induction hypothesis.

In the remaining two cases we assume that \( P \) is of the form \( f(l_1, \ldots, l_m) \rightarrow r \ll F^* \).

(2.2) Suppose \( s \) is of the form \( f(s_1, \ldots, s_m) \). Let \( F = F' \cup F^* \cup \{ s_i \phi l | 1 \leq i \leq m \} \cup \{ r \equiv t \} \). Then,

\[
F' \cup \{ s \phi t \} \rightarrow \psi \phi F.
\]

By an application of Proposition 5.1 to (2) we find a refutation of \( EP \cup \{ \equiv F \} \) with respect to \( \{ \rightarrow_{ip}, \rightarrow_{r} \} \), computed answer substitution \( \theta = \theta^* \), and complexity \( M \ll M^* \ll M' \). The result follows by the induction hypothesis.

(2.3) Suppose \( s \) is a variable. Hence, \( t \) must be a non-variable term. Let \( x_i, 1 \leq i \leq m \) be new variables, \( \gamma = \{ s \phi f(x_1, \ldots, x_m) \} \), and \( F = \gamma(F' \cup F^* \cup \{ x_i \phi l | 1 \leq i \leq m \} \cup \{ r \equiv t \}) \). Then,

\[
F' \cup \{ s \phi t \} \rightarrow \varphi \phi F.
\]

By an application of Proposition 5.3 to (2) we find

\[
\gamma(F' \cup F^* \cup \{ s \phi l, r \equiv t \})
\]

\[
= \gamma F' \cup F^* \cup \{ f(x_1, \ldots, x_m) \phi f(l_1, \ldots, l_m), r \equiv \gamma t \}
\]

\[
\equiv \square
\]

(3) Suppose that in (1) paramodulation is never applied to an element of a descendant of \( s \phi t \).

(3.1) If reflection is applied in the first step of (1) then the result follows in analogy to case 1.

In the remaining two cases we may assume that instantiation or paramodulation (using \( P = l \rightarrow r \ll F^* \)) is applied in the first step of (1). Recall, \( s \) and \( t \) cannot both be variables.

(3.2) Suppose \( s \) (resp. \( t \)) is of the form \( f(s_1, \ldots, s_m) \) (resp. \( g(t_1, \ldots, t_n) \)). Since in (1) paramodulation is never applied to an element of a descendant of \( s \phi t \) we find that \( f = g \) and \( n = m \). Let \( F = F' \cup \{ s_i \phi t_i | 1 \leq i \leq m \} \). Then,

\[
F' \cup \{ s \phi t \} \rightarrow \phi F
\]

and by an application of Proposition 5.1 to (1) we find a refutation of \( EP \cup \{ \equiv F \} \) with respect to \( \{ \rightarrow_{ip}, \rightarrow_{r} \} \), computed answer substitution \( \theta = \theta^* \), and complexity \( M \ll M' \). The result follows by the induction hypothesis.
(3.3) Finally, suppose that $s$ is a variable and $t$ is a term of the form $f(t_1, \ldots, t_m)$. Since in (1) paramodulation is never applied to an element of a descendant of $s \equiv t$ we find a binding $s \leftarrow f(s_1, \ldots, s_m)$ in $\theta'$. Now let $x_i, 1 \leq i \leq m$, be new variables, $\gamma = \{x \leftarrow f(x_1, \ldots, x_m)\}$, and $F = \gamma(F' \cup \{x_i \equiv t_i \mid 1 \leq i \leq m\})$. Then,

$$\iff F' \cup \{s \equiv t\} \rightarrow_{im} F.$$  

By an application of Propositions 5.3 and 5.1 to (1) we find a refutation of $EP \cup \{\iff F\}$ with respect to $\{\rightarrow_{ip}, \rightarrow_{r}\}$, computed answer substitution $\theta$, and complexity $M$ such that $\theta \gamma|_{\text{Var}(F' \cup \{x \equiv t\})}$ is more general than $\theta'$ and $M \leq M'$. The result follows by the induction hypothesis. □

The proof of Theorem 5.5 gives us a procedure that transforms refutations with respect to reflection, instantiation and paramodulation into refutations with respect to $TRANS$, e.g. this procedure transforms the refutation in Fig. 1 into the refutation depicted in Fig. 2.

$$\iff f(x, x) \equiv d(x, x) \rightarrow_{ln(f)} \iff x \equiv c(g), x \equiv c(a), d(x, x) \equiv d(c(g), c(a))$$  

$$\rightarrow_{x \leftarrow \{c(g)\}} \iff c(g) \equiv c(a), d(c(g), c(g)) \equiv d(c(g), c(a))$$  

$$\rightarrow_d \iff g \equiv a, d(c(g), c(g)) \equiv d(c(g), c(a))$$  

$$\rightarrow_d \iff g \equiv a, c(g) \equiv c(g), c(g) \equiv c(a)$$  

$$\rightarrow_t \iff g \equiv a, c(g) \equiv c(a)$$  

$$\rightarrow_d \iff g \equiv a, g \equiv a$$  

$$\rightarrow_{ln(g)} \iff a \equiv a, g \equiv a$$  

$$\rightarrow_t \iff g \equiv a$$  

$$\rightarrow_{ln(g)} \iff a \equiv a$$  

$$\rightarrow_t \iff a \equiv a$$

Fig. 2.

It should be noted that the empty clause is derived in Fig. 2 by applying only lazy narrowing, term decomposition, variable elimination, and removal of trivial equations. This is remarkable since the FUN-example has served to show that paramodulation and reflection is complete only if an instantiation rule is added. Since lazy narrowing is applied to $f(x, x)$ using $(f)$, the uninformed use of the instantiation rule in the refutation of $FUN \cup \{\iff f(x, x) \equiv d(x, x)\}$ with respect to $\{\rightarrow_{ip}, \rightarrow_{r}\}$ to instantiate the variable $x$ is replaced by an informed application of term decomposition in the corresponding refutation of $FUN \cup \{\iff f(x, x) \equiv d(x, x)\}$ with respect to $TRANS$. 

We can now show that the transformations are complete.

**Theorem 5.6 (Strong Completeness of TRANS).** For every correct answer substitution \( \theta \) for EP and \( \leftarrow F \) there exists an \( R^+ \)-computed answer substitution \( \sigma \) obtained by a refutation of \( EP \cup EP^{-1} \cup \{ \leftarrow F \} \) with respect to TRANS such that \( \sigma \geq_{EP} \theta \).

**Proof.** If \( \theta \) is a correct answer substitution for EP and \( \leftarrow F \), then we find a computed answer substitution \( \gamma \) obtained by a refutation of \( EP \cup EP^{-1} \cup \{ \leftarrow F \} \) with respect to \( \{ \rightarrow_{ip}, \rightarrow_{r} \} \) such that \( \gamma \geq \theta \) (Theorem 3.1). The result follows immediately by an application of Theorem 5.5. \( \Box \)

6. Refining the transformations

The proof of the completeness of the transformation rules suggests that the refinements of paramodulation can be carried over to refutations with respect to TRANS. In Theorem 5.5 we have shown that for each refutation of

\[
EP \cup \{ \leftarrow F \} \quad \text{with respect to} \quad \{ \rightarrow_{ip}, \rightarrow_{r} \} \tag{4}
\]

and computed answer substitution \( \theta \) we can find a refutation of

\[
EP \cup \{ \leftarrow F \} \quad \text{with respect to} \quad TRANS \tag{5}
\]

yielding a computed answer substitution \( \sigma \) such that \( \sigma \) is more general modulo EP than \( \theta \). If we take a close look at the proof of this theorem we can make the following observations.

An equational clause \( P \) is used in an \( \rightarrow_{ln}, \rightarrow_{pv}, \) or \( \rightarrow_{tc} \) step in (5) only if the same clause is used in a paramodulation step in (4). Moreover, in both refutations \( P \) is used in the same direction. Hence, by Theorem 3.4 clauses from \( EP^{-1} \) are no longer needed if EP is ground confluent.

**Corollary 6.1.** Let EP be a ground confluent equational program. For every computed answer substitution \( \theta \) for EP and \( \leftarrow F \) there exists an \( R^+ \)-computed answer substitution \( \sigma \) obtained by a refutation of \( EP \cup \{ \leftarrow F \} \) with respect to TRANS such that \( \sigma \geq_{EP} \theta \).

If paramodulation is applied to a variable in (5), then the computed answer substitution is not in normal form. As an example consider the equational clause \( f(x) \rightarrow b \leftarrow \) and the equation \( y \equiv b \).

\[
\leftarrow y \equiv b \rightarrow_{pv(y \leftarrow f(x))} \leftarrow z \equiv x, \quad b \equiv b \rightarrow, \quad \leftarrow z \equiv x \rightarrow v(z \leftarrow y)) \quad \Box
\]

with computed answer substitution \( \{ y \leftarrow f(x) \} \). Furthermore, a trivial clause is applied in (5) only if the same trivial clause is applied in (4). Hence, if EP is non-trivial, then the rule \( \rightarrow_{tc} \) is no longer needed.
Now, since narrowing and reflection is complete for non-trivial and ground confluent term rewriting systems as long as we consider only normalized answer substitutions (Theorem 3.5), we conclude that in this case \( \rightarrow_{pv} \) and \( \rightarrow_{tc} \) are unnecessary and that we have not to restrict our computation rule.

**Corollary 6.2.** Let \( R \) be a computation rule and \( EP \) be a non-trivial and ground confluent term rewriting system. For every normalized correct answer substitution \( \theta \) for \( EP \) and \( F \) there exists an \( R \)-computed answer substitution \( \sigma \) obtained by a refutation of \( EP \cup \{F\} \) with respect to \( \{\rightarrow_d, \rightarrow_v, \rightarrow_{tn}, \rightarrow_{in}\} \) such that \( \sigma \geq_{EP} \theta \).

Finally, if the equational program is a canonical term rewriting system, then the rules removal of trivial equations, decomposition of decomposable equations, variable elimination applied to equations of the form \( x = t \), where no defined function symbol occurs in \( t \), and rewriting can be applied as simplification rules to refutations with respect to narrowing and reflection. Furthermore, there does not exist a refutation of \( EP \cup \{F\} \) with respect to \( TRANS \) if \( F \) contains an equation of the form \( c(s_1, \ldots, s_n) \equiv d(t_1, \ldots, t_m) \), where \( c \) and \( d \) are different constructors. Such a goal clause is often called a failure.

In analogy to [28] we define a function \( simplify \) which applies the above mentioned simplification rules to a goal clause as long as possible and tests that it is not a failure. An \( s \)-derivation is a derivation where each goal clause is simplified.

**Theorem 6.3.** Let \( R \) be a computation rule, \( EP \) be a canonical term rewriting system, and \( \theta \) be a normalized correct answer substitution for \( EP \) and \( F \). Then there exists an \( R \)-computed answer substitution \( \sigma \) obtained by an \( s \)-refutation of \( EP \cup \{F\} \) with respect to \( \{\rightarrow_d, \rightarrow_v, \rightarrow_{tn}, \rightarrow_{in}\} \) such that \( \sigma \geq_{EP} \theta \).

**Proof.** The proof is in analogy to the proof of Theorems 5.5 and 5.6 except that the first component \( \#p \) of the complexity used in the proof of Theorem 5.5 must be the maximum number of applications of rewriting steps in the refutation of \( EP \cup \{F\} \) with respect to rewriting and removal of trivial equations. \( \square \)

To example the notion of an \( s \)-refutation and its advantages consider the rewrite rules:

\[
\begin{align*}
\text{append}([\ ] \ , z) & \rightarrow z \leftarrow \quad (a1) \\
\text{append}(x : y \ , z) & \rightarrow x : \text{append}(y \ , z) \leftarrow \quad (a2) \\
\text{map-succ}([\ ] \ ) & \rightarrow [\ ] \leftarrow \quad (m1) \\
\text{map-succ}(x : y \ ) & \rightarrow \text{succ}(x) : \text{map-succ}(y) \leftarrow \quad (m2)
\end{align*}
\]

Informally, the function \( \text{append} \) appends two lists built from the constant \( [\ ] \) denoting the empty list and the list-constructor \( : \). The function \( \text{map-succ} \) applied...
to a list of natural numbers adds 1 to each element of a list. It is easy to see that there exists an infinite derivation of

\[ \Leftarrow \text{map-succ}(0 : \text{append}(x, y)) \Rightarrow 0 : z \]

with respect to narrowing and reflection by repeatedly applying narrowing and using (a2):

\[ \Rightarrow \text{map-succ}(0 : \text{append}(x, y)) \Rightarrow 0 : z \]

\[ \rightarrow_n \Rightarrow \text{map-succ}(0 : z' : \text{append}(y', z')) \Rightarrow 0 : z \]

\[ \rightarrow_n \cdots . \]

Note that we have applied narrowing always to innermost basic occurrences (see e.g. [39]). However, the initial goal clause can be simplified by rewriting

\[ \text{map-succ}(0 : \text{append}(x, y)) \]

to

\[ \text{succ}(0) : \text{map-succ}(\text{append}(x, y)) \]

using (m2). Since \text{succ} and 0 are different constructors we conclude that this goal clause cannot be solved and terminate the derivation after a single simplification step.

We finish this section by showing that the imitation rule is needed to ensure Theorem 6.3. Let \{f(x) \rightarrow a \subseteq \} be the canonical term rewriting system. Then,

\[ \Leftarrow y \doteq c(f(y)) \rightarrow_{\text{im}(\{y \leftarrow c(x)\}))} \Leftarrow z \doteq f(c(z)) \]

\[ \rightarrow_{\text{in}} \Leftarrow z \doteq a, c(z) \doteq x \]

\[ \rightarrow_{\nu(\{z \leftarrow a\})} \Leftarrow c(a) \doteq x \]

\[ \rightarrow_{\nu(\{x \leftarrow c(a)\})} \]

with computed answer substitution \{y \leftarrow c(a)\}. It should be noted that imitation is the only inference rule which is applicable to \( y \doteq c(f(y)) \).

7. Discussion

We have generalized results obtained by Gallier and Snyder [20, 21] and Martelli et al. [50] to hold for arbitrary equational programs (resp. conditional term rewriting systems). Moreover, we have refined their results. To ensure the completeness of their sets of transformations for canonical term rewriting system, Gallier and Snyder and Martelli et al. have modified the lazy narrowing rule to be applicable also to arbitrary proper subterms of an equation. This not only violates the demand driven nature of the transformation rules but also expands the search space since there are
generally several subterms of an equation to which their lazy narrowing rule can be applied. We ensure the completeness by repeated applications of the imitation rule as shown in the last example of Section 6.

On the other hand, since we have integrated the application of simplification rules, we can sometimes reduce an infinite search space with respect to the rules of Gallier and Snyder to a finite one, as the *append*-example in the previous section shows.

Gallier and Snyder [20, 21] have pointed out that successive applications of the imitation rule to an equation of the form \( x \equiv t \), where \( x \) occurs in \( t \), will generate an instance of the equation and, thus, lead to a cycle. However, they have also shown that in the case of unconditional equational theories these cycles can be avoided. We believe that this result holds also for Horn equality theories.

There is another important difference between Gallier's and Snyder's [21] approach and the one presented herein. Using a simplification ordering \( \succ \) which is total on ground terms and an unfailing completion procedure (e.g. [34]), they show that there exists a ground confluent completion \( \text{EP}^\omega \) for an arbitrary unconditional equational theory \( \text{EP} \). Since there is no loss in generality in considering ground substitutions and a ground substitution can be normalized with respect to \( \succ \), they conclude that paramodulation upon variables is unnecessary using similar arguments which led to our Corollary 6.2. Finally, they prove that the unfailing completion procedure can be simulated by the transformation rules without applying paramodulation upon variables. This leads generally to a considerable restriction of the search space. On the other hand, it may be necessary to apply clause \( \text{EP}^{-1} \) even if \( \text{EP} \) is ground confluent as we have shown in Section 4. Since the unfailing completion procedure can be generalized to conditional equational theories it should be possible to extend Gallier's and Snyder's results to conditional equational theories as well.

It should be observed that, if variable elimination can be applied as a simplification rule, the transformation rules can be refined considerably: imitation and paramodulation to variables need only to be applied to \( x \equiv t \) if \( x \) occurs in \( t \). Similarly, lazy narrowing and application of a trivial clause need not to be applied to \( x \equiv t \) if \( x \) does not occur in \( t \). Though many researchers have suggested the use of variable elimination as a simplification rule [20, 29, 50], none of them has been able to give a rigorous proof for it. Only recently Hsiang and Jouannaud [33] have announced such a proof for unconditional theories.

In this paper we consider only first-order equational theories. Snyder and Gallier [22] have defined a complete higher-order unification procedure based on sets of transformations. Moreover, Gallier et al. [23] have extended this result to hold also for higher-order E-unification.

The transformation rules presented herein can be used as a computational method for equational logic programs as proposed by Jaffar et al. [41, 47] or Goguen and Meseguer [25] by adding a lazy resolution rule as suggested in [28]. This rule applied to a selected atom of the form \( P(s_1, \ldots, s_n) \) and a program clause of the form
Let $P(t_1, \ldots, t_n) \leftarrow D^*$ forces the comparison of corresponding arguments, i.e.

$$
\leftarrow D \cup \{P(s_1, \ldots, s_n)\} \rightarrow \leftarrow D \cup D^* \cup \{s_i \vdash t_i | 1 \leq i \leq n\},
$$

where $D$ and $D^*$ are sets of atoms and equations.

There are, of course, other proposals to handle equational theories. We have already mentioned paramodulation and special forms of it such as narrowing (e.g. [24, 39, 40, 44, 58, 59]) or superposition [14, 15]. It seems that the use of transformation rules cuts down the search space since there are less alternatives, the application is demand driven, and failures can be recognized earlier.

Another proposal is based on the idea to flatten goal and program clauses and then to apply SLD-resolution (e.g. [3, 8, 10]). The disadvantage of this technique is that rewriting can no longer be applied as a simplification rule. It can only be simulated by a sequence of SLD-resolution steps using a complex computation rule. However, rewriting goal clauses may cut the search space from an infinite to a finite one. Recently, Nutt et al. [53] have shown that narrowing and flattening can be combined in one system leaving it to the overall strategy whether goal clauses should be flattened or narrowing should be applied.

Unfortunately, there has been no thorough comparison between the various techniques so far. We only know for sure that each of them is superior to the others in certain aspects or for certain classes of equational theories.

**References**


Complete sets of transformations


