# INDUCTIVE INFERENCE FOR SOLVING DIVERGENCE IN KNUTH-BENDIX COMPLETION

Muffy Thomas
University of Glasgow, Dept. of Computing Science, Lilybank Gardens, Glasgow, G12, Scotland.

Klaus P. Jantke
Leipzig University of Technology, Dept. of Mathematics & Informatics, P.O. Box 66, 7030 Leipzig, German Democratic Republic.

## 1. INTRODUCTION

This paper presents an approach to solving divergence in the *Knuth-Bendix completion algorithm* [Knuth/Bendix].

The Knuth-Bendix completion procedure generates a *confluent* set of rewrite rules by repeatedly superposing left hand sides of rewrite rules and adding any generated critical pairs as new rewrite rules. The process may terminate in two ways; with success: there are no more (non-trivial) critical pairs, or with failure: a critical pair is generated which cannot be oriented by the ordering. The process is said to *diverge* if it does not terminate at all.

When the confluent set is infinite (i.e. an infinite sequence of critical pairs is generated), then we attempt to replace the infinite sequence of rewrite rules by a finite sequence of rules which are equivalent in some sense. Obviously, this is only possible if the word problem under consideration is decidable. Therefore, the approach presented here is only applicable in an enumerable number of cases. We note that our approach differs from that in [Kirchener] where *meta*-variables and *meta*-rules are introduced for solving divergence problems; our aim is to derive an ordinary, finite, rewrite rule system which is confluent and terminating.

The key idea of our approach consists of invoking inductive inference techniques (cf. the survey in [Angluin/Smith]) when generalising a given sequence in order to replace it by a single term.

The basic notions and notations of algebraic specification and term rewriting systems from [Ehrig/Mahr, Huet/Oppen] are assumed. The paper is organised in the following way. In sections 2 and 3 the basic definitions of rewriting and generalisation are given. Section 4 contains a brief introduction to inductive inference. Section 5 contains the main example which illustrates the divergence problem and section 6 contains two inductive inference algorithms which may be applied to the example. In section 7 the conclusions and directions for future work are discussed.

## 2. TERM REWRITING AND CONFLUENCE

We assume a finite, heterogeneous signature $\Sigma$ with an appropriate set of sorted variables such that every sort contains at least one ground term, i.e. carrier sets in the corresponding algebras are not empty. Terms are defined as usual; terms which do not contain variables are called *ground* terms.

A given rewrite rule system **R** (over $\Sigma$) defines the rewrite relation $\rightarrow_R$, the transitive closure $\rightarrow^*_R$, and the equivalence relation $\leftrightarrow^*_R$ as usual.

Confluent and terminating (i.e. *canonical*) term rewriting systems are desirable because they ensure that every term has a unique normal form and thus equality between terms is decided by comparing normal forms. Moreover, (finite) canonical rewriting systems provide an operational semantics for abstract data types with initial algebra semantics.

When a given system is not confluent, one can attempt to complete it by means of a completion algorithm such as the Knuth-Bendix algorithm. This algorithm computes critical expressions and critical pairs which are then proposed as new rules. We assume that it is possible to orient a critical pair as desired (w.r.t. an underlying well-founded ordering on terms) such that the rule may be inserted into the given rewrite rule system without violating termination; we do not consider termination orderings explicitly here. Moreover, we assume that all terms which occur in a new rewrite rule are in normal form.

When the process diverges and we have an *infinite* set of confluent rewrite rules, then we only have a semi-decision procedure. Consider the situation where an application of a critical pair completion procedure to a given rewrite rule system **R** results in an infinite sequence of critical pairs $<l_1,r_1>$, $<l_2,r_2>$, $<l_3,r_3>$, . . . which should be considered as new rewrite rules $l_1 \rightarrow r_1$, $l_2 \rightarrow r_2$, $l_3 \rightarrow r_3$, . . . For a given rewrite rule system **R**, we denote the (possibly) infinite set of (persistent) canonical rules generated by the completion procedure by $R^\infty$. If this is an infinite sequence and to be replaced by a finite set of rules, such a replacement, an *enrichment* of **R**, should meet certain conditions. First, the enrichment should be canonical and second, it should preserve the equational theory defined by **R** (or the rewrite relation defined by $R^\infty$). The enrichment of **R**, denoted here by **R'**, may be based on a larger signature; i.e. the new rules in **R'** may use some new operators and, perhaps, some new sorts. Intuitively, the enrichment **R'** should reflect exactly the equational theory of **R**, when restricted to the original sorts. This notion is referred to as ground consistency, or a conservative extension [Ehrig/Mahr]:

**Definition**
Let **R** be a rewrite rule system over a given signature $\Sigma$ and let **R**$'$ be a finite enrichment of **R**. We say that **R'** is a *conservative extension* of **R** when

$$(\forall t_1, t_2 \in T_\Sigma) \;\; t_1 \leftrightarrow^*_{\mathbf{R}} t_2 \quad \text{iff} \quad t_1 \leftrightarrow^*_{\mathbf{R}'} t_2 \;.$$

This definition reflects our intention that a correct completion must not imply any new equalities among ground terms over the original sorts. We note that we do not require this condition for terms which contain variables; namely, we may introduce theorems from the inductive theory.

## 3. GENERALISATION

In this section we present some basic generalisation concepts. Our motivation is that a given sequence of terms $\{t_1, t_2, ...\}$ may be generalised, in some sense, by a single term. The concepts should reflect our intuition as well as being computationally tractable. We note that although the definitions below apply to sequences of terms, by an abuse of notation they also apply sequences of rewrite rules when we consider a rewrite rule $l \rightarrow r$ as a term with $\rightarrow$ as its outermost operator.

When the terms occurring in a sequence $\{t_1, t_2, ...\}$ contain variables, then our intuitive notion of generalising this sequence must take into account all possible ground instances of all the individual terms. In order to formalise this, we define the *language* generated by a term, or a set of terms, respectively.

**Definition**
Let R be a terminating rewrite rule system over signature $\Sigma$ and let X be a set of sorted variables. Let t be a term in $T_\Sigma(X)$ and let T be a subset of $T_\Sigma(X)$. We define the *language of t*, $L(t)$ and the *language of T*, $L(T)$, by

$$L(t) \quad =_{\text{def}} \; \{t' \mid (\exists \sigma: X \rightarrow T_\Sigma) \; t\sigma = t'\}$$
$$L(T) \quad =_{\text{def}} \; \text{the union of all } L(t) \text{ for } t \in T$$

We also define the *normal form* language (w.r.t. a rewrite rule system) of a term, or set of terms.

**Definition**
Let R be a terminating rewrite rule system over signature $\Sigma$. For each set $M$ of terms in $T_\Sigma$, $\downarrow M$ denotes the set of all normal forms of terms in $M$, w.r.t. $R^\infty$.

Now, two generalisation relations, $>>$ and $>>_N$, are defined.

**Definition**
Let R be a rewrite rule system over signature $\Sigma$, and let $t, t_1, t_2, \ldots$ be terms in $T_\Sigma(X)$.

$$t >> \{t_1, t_2, \ldots\} \quad \text{iff } L(t) \supseteq L(\{t_1, t_2, \ldots\}).$$

$$t >>_N \{t_1, t_2, \ldots\} \quad \text{iff } \downarrow L(t) \supseteq \downarrow L(\{t_1, t_2, \ldots\}).$$

When $L(t) = L(\{t_1, t_2, \ldots\})$, then t is called an *exact generalisation* of $\{t_1, t_2, \ldots\}$.
When $\downarrow L(t) = \downarrow L(\{t_1, t_2, \ldots\})$, then t is called an *exact normal generalisation* of $\{t_1, t_2, \ldots\}$.

Clearly, enriching a given rewrite rule system **R** with an exact generalisation will result in a conservative extension of **R**. Moreover, enriching **R** with an exact *normal* generalisation also results in a conservative extension of R.

**Lemma**
Let **R** be a given rewrite rule system and let $\rho$ be an exact normal generalisation of $R^\infty$. $R \cup \rho$ is a conservative extension of **R**.

**Proof**
The implication from left to right is trivial; we prove the converse by contradiction. Assume that $\rho$ implies a new equivalence between ground terms $s_1$ and $s_2$; namely, $s_1 \rightarrow s_2$ is a ground instance of $\rho$, $s_1 \leftrightarrow^*_{R \cup \rho} s_2$ but $s_1 \leftrightarrow^*_R s_2$ does not hold. Clearly $\downarrow(\rho) \supseteq \downarrow(\{s_1 \rightarrow s_2\})$ w.r.t. $R \cup \rho$. However, because $\rho$ is an exact normal generalisation, then we also have $\downarrow(\rho) \supseteq \downarrow(\{s_1 \rightarrow s_2\})$ w.r.t. R and as a consequence $s_1 \leftrightarrow^*_R s_2$ which contradicts the assumption.

Although the above notions of generalisation are very intuitive, they are not computationally tractable. Therefore, we introduce some further definitions.

**Definition**
Let **R** be a rewrite rule system over signature $\Sigma$, let $t, t_1, t_2, \ldots$ be terms in $T_\Sigma(X)$ and let $\sigma_1, \sigma_2, \ldots$ be substitutions from X to$T_\Sigma(X)$.

| | | |
|---|---|---|
| $t > \{t_1, t_2, \ldots\}$ | iff | For every term $t_i$ there is a substitution $\sigma_i$ such that $t\sigma_i = t_i$. |
| $t > \{t_1, t_2, \ldots\}(\rightarrow^*_R)$ | iff | For every term $t_i$ there is a substitution $\sigma_i$ such that $t\sigma_i \rightarrow^*_R t_i$. |
| $t > \{t_1, t_2, \ldots\}(\leftrightarrow^*_R)$ | iff | For every term $t_i$ there is a substitution $\sigma_i$ such that $t\sigma_i \leftrightarrow^*_R t_i$. |

The relation > is also referred to as the *subsumption* relation and is used as the generalisation concept in [Kodratoff]; the relation > (modulo any theory) is used as the generalisation concept in [Plotkin]. We note that these definitions are not comprehensive and that there are some more possibilities, e.g. $\rightarrow^*_R$ may be replaced by $\leftarrow^*_R$.

The required properties are difficult to verify, in general, except the condition of $>$, (with the empty theory), which is purely syntactical and decidable. This is a good reason to prefer the relation $>$ as a concept of generalisation. The relation $>(\leftrightarrow^*_R)$ causes serious problems as it has been shown [Lange] that it is undecidable whether or not there exists a generalisation (modulo an equivalence) of two arbitrary terms $\{t_1, t_2\}$.

**Proposition**

1. $t > \{t_1, t_2, ...\} \quad \Rightarrow \quad t \quad > \quad \{t_1, t_2, ...\} (\rightarrow^*_R)$
2. $t > \{t_1, t_2, ...\}(\rightarrow^*_R) \quad \Rightarrow \quad t \quad > \quad \{t_1, t_2, ...\} (\leftrightarrow^*_R)$
3. $t > \{t_1, t_2, ...\} \quad \Rightarrow \quad t \quad >> \quad \{t_1, t_2, ...\}$
4. $t > \{t_1, t_2, ...\}(\rightarrow^*_{R^\infty}) \Rightarrow \quad t \quad >>_N \{t_1, t_2, ...\}$

**Proof of 4**

If $t > \{t_1, t_2, ...\}(\rightarrow^*_{R^\infty})$, then by definition of $>$, for each $t_i$, $i=1,2,...$, there exists an $\sigma_i$ s.t. $t\sigma_i \rightarrow^*_{R^\infty} t_i$. By definition of normal forms, for each $t_i$ there is a normal form $\downarrow t_i$ s.t. $t_i \rightarrow^*_R \downarrow t_i$. If for some t, we have $t\sigma_i \rightarrow^*_{R^\infty}$ t, then by the confluence property, we also have $t \rightarrow^*_{R^\infty} \downarrow t_i$. Thus, for every $\downarrow t_i$ there is a substitution $\sigma_i$ s.t. $\downarrow t\sigma_i = \downarrow t_i$.

**Lemma**

$\neg(t >> \{t_1, t_2, ...\} \Rightarrow t > \{t_1, t_2, ...\})$.

*Proof*: By counter example. Assume any signature with at least three sorts $s$, $s'$, $s''$, an operator $h: s \rightarrow s''$, exactly one s-sorted operator $f: s' \rightarrow s$ and there are no constants of sort s. Let x and y be variables of sort s and $s'$ respectively. Now consider the following two terms: $t =_{def} h(f(y))$ and $t' =_{def} h(x)$. Because all instantiations of h(x) must have the form h(f(z)), where z has sort $s'$, both terms define the same language of ground terms; i.e. $t >> \{t'\}$ and $t' >> \{t\}$. But clearly $t > \{t'\}$ does not hold.

**Lemma**

If every sort contains at least one constant and two distinct ground terms, then
$t >> \{t_1, t_2, ...\} \Rightarrow t > \{t_1, t_2, ...\}$.

*Proof*: Let $t, t_1, t_2$ ... be terms with $t >> \{t_1, t_2, ...\}$. Consider any term $t' \in \{t_1, t_2, ...\}$ and let $\sigma$ be a ground substitution which substitutes every variable in t' by a constant. The resulting term $t'\sigma$ has the same structure t' and, by assumption, must belong to $L(t)$. Thus, t' cannot be structurally more general than t: at the occurrences of variables in t', t must contain either variables or constants. If, at such a position in t, there is a constant, then one could generate from t' a term which could not be generated from t; (this is possible by substituting another ground term which exists by assumption). Thus, at every occurrence of a variable in t', t must also have a variable at this position. Moreover, distinct variables in t' must also be distinct in t. Otherwise, by using the two different ground terms, one could generate a term from t' which cannot be generated from t. Thus, t must be identical to or structurally more general than the chosen term t'.

Finally, we note that must of course exclude certain "unacceptable" generalisations which are formally possible, but generate rewrite rules which are unacceptable: e.g. $x \to y$. In the following, we call a rewrite rule $l \to r$ *acceptable*, if all variables occurring in r also occur in l.

Our aim is to find, where possible, ground consistent, acceptable generalisations of sequences of critical pairs. Fortunately, we see that the simple, syntactic notion of $>$ is equivalent in many cases. Moreover, we shall look for generalisations which are minimal, exact or exact normal generalisations.

## 4. INDUCTIVE INFERENCE

Inductive inference [Angluin/Smith] is a mathematical theory of algorithmic learning from incomplete information. As inductive inference addresses the problem of learning from *incomplete* information, when applying an inductive inference procedure to some learning problem, one never knows in advance whether or not the information processed so far is comprehensive enough to construct reasonable hypotheses about the phenomenon to be learned. Therefore, even successful inductive inference methods may offer only a semi-computable tool. Fortunately, this fits very well into the framework of critical pair completion procedures where termination (of the algorithm) is undecidable.

In general, it is impossible to prove the correctness of an hypothesis with respect to some infinite sequence of information as each hypothesis is usually generated from a finite, initial segment of the sequence. An inductive inference strategy has to be designed such that its result becomes correct after processing a sufficiently large, but still finite, amount of information. Every wrong hypothesis must be changed after a finite number of steps.

For example, if there is an sequence of critical pairs/rewrite rules:

$$l_1 \to r_1,\ l_2 \to r_2,\ l_3 \to r_3,\ \ldots$$

and if some generalised rule $l \to r$ has been constructed from the initial part

$$l_1 \to r_1, \ldots, l_n \to r_n,$$

then it is impossible to prove that this rule is a generalisation of the whole sequence. Consequently, there is no way to prove that a generalisation is minimal, or exact, in general. However, this may be possible in particular cases using additional knowledge about the sequence of critical pairs, e.g. by describing (finitely) how the sequence is generated.

We will use inductive inference when constructing a generalisation of a sequence from a finite initial segment; in the following section we give an example in order to motivate the generalisation algorithms.

## 5. AN EXAMPLE

Considering the following rewrite rule system **R**. The signature contains the sort **nat** with the usual arithmetic operators 0:**nat**, S: **nat** -> **nat** and _+_: **nat nat** -> **nat**, and the sort **T** with operators f, g, h have arities **T** -> **T**, **T** -> **T**, and **nat T** -> **T**,resp.

| | | | |
|---|---|---|---|
| (A1) | x+0 | → | x |
| (A2) | 0+x | → | x |
| (A3) | x + S(y) | → | S(x + y) |
| (A4) | S(x) + y | → | S(x + y) |
| (A5) | (x+y)+ z | → | x + (y + z) |

| | | | |
|---|---|---|---|
| (B1) | f(g(f(x))) | → | f(h(S(0),x)) |
| (B2) | f(g(h(y,x))) | → | f(h(S(y),x)) |
| (B3) | f(h(z,h(y,x))) | → | f(h(z + y,x)) |

The subsystem consisting of **(A1), ... ,(A5)** is obviously canonical, but the system as a whole is not confluent. The generated sequence of critical pairs/rewrite rules is:

| | | | |
|---|---|---|---|
| (C1) | f(h(S(0),g(f(x)))) | → | f(h(S(S(0)),x)) |
| (C2) | f(h(S(0),g(h(y,x)))) | → | f(h(S(S(y)),x)) |
| (C3) | f(h(S(S(0)),g(f(x)))) | → | f(h(S(S(S(0))),x)) |
| (C4) | f(h(S(S(0)),g(h(y,x)))) | → | f(h(S(S(S(y))),x)) |
| (C5) | f(h(S(S(S(0))),g(f(x)))) | → | f(h(S(S(S(S(0)))),x)) |
| (C6) | f(h(S(S(S(0))),g(h(y,x)))) | → | f(h(S(S(S(S(y)))),x)) |
| (C7) | f(h(S(S(S(S(0)))),g(f(x)))) | → | f(h(S(S(S(S(S(0))))),x)) etc. |

For generalisation, the sequence is partitioned into two subsequences: **(C1), (C3), (C5),** ... and **(C2), (C4), (C6),** ...

| | | | |
|---|---|---|---|
| (C1) | f(h(S(0),g(f(x)))) | → | f(h(S(S(0)),x)) |
| (C3) | f(h(S(S(0)),g(f(x)))) | → | f(h(S(S(S(0))),x)) |
| (C5) | f(h(S(S(S(0))),g(f(x)))) | → | f(h(S(S(S(S(0)))),x)) |
| (C7) | f(h(S(S(S(S(0)))),g(f(x)))) | → | f(h(S(S(S(S(S(0))))),x)) etc. |

| | | | |
|---|---|---|---|
| **(C2)** | f(h(S(0),g(h(y,x)))) | $\rightarrow$ | f(h(S(S(y)),x)) |
| **(C4)** | f(h(S(S(0)),g(h(y,x)))) | $\rightarrow$ | f(h(S(S(S(y))),x)) |
| **(C6)** | f(h(S(S(S(0))),g(h(y,x)))) | $\rightarrow$ | f(h(S(S(S(S(y)))),x)) etc. |

The motivation for the partitioning is that we wish to generalise every resulting (sub)sequence w.r.t. one of the generalisation concepts introduced above: every sequence should be replaced by a single rule. The question of how, in general, to partition a sequence into appropriate subsequences is a difficult problem and remains open. For the moment, there are some useful heuristics. For a given critical pair, call the (superposed) rules which generate it its *parents*. Now, every rule in the first sequence has the rule **(B1)** among its parents, whereas every rule of the second sequence has **(B2)**. If we consider, additionally, grandparents and so on, this may provide sufficient information for partitioning a given sequence of critical pairs into generalisable subsequences.

Before introducing the algorithms for generalising rules, we use the example for illustration.

For the first sequence: **(C1), (C3), (C5),** ... , the rule

**(G1)** f(h(S(y),g(f(x)))) $\rightarrow$ f(h(S(S(y)),x))

is a generalisation of the sequence **(C1), (C3), (C5),**... w.r.t. >. (Recall that rewrite rules are considered as terms with $\rightarrow$ as outermost operator.) Moreover, **(G1)** is a generalisation of the sequence **(C1), (C3), (C5),** ... w.r.t. all the other generalisation concepts. **(G1)** is not an exact generalisation of **(C1), (C3), (C5),** but it is an exact normal generalisation and therefore it is a conservative extension of **R1**.

Generalising the second sequence **(C2), (C4), (C6),** ... w.r.t. > is more difficult. In particular, there is no exact or exact normal generalisation which is a generalisation w.r.t. >. However,

**(G2)** f(h(S(z),g(h(y,x)))) $\rightarrow$ f(h(S(S(z+y)),x))

is a generalisation w.r.t. $>(\rightarrow^*_{R^\infty})$ and and it is an *exact* normal generalisation. Namely, after substituting any ground term of sort nat for z in **(G2)**, the resulting subterm is always rewritten into a normal form which occurs in a particular rule **(Ci)**. For example, substituting (S(0)+0)+S(S(0)) for **z** yields the normal forms S(S(S(S(0)))) and S(S(S(S(S(y))))) which are the left and right hands (resp.) of the rule **(C8)** .

To summarise, **(A1),** ... **,(A5), (G1) (G2)** is a canonical rewrite system and {**(G1), (G2)**} is a conservative extension of **(A1),** ... **,(A5).**

## 6. TWO BASIC INDUCTIVE INFERENCE ALGORITHMS

In this section we give two inductive inference algorithms for generalising infinite sequences of critical pairs. Each algorithm takes as input some finite, initial segment of such a sequence and generates single rules intended to be generalisations. Because generalisation is associative, we assume exactly two examples as input: $l_1 \to r_1$ and $l_2 \to r_2$.

In the following, we use the convention that when p is a *position* in a term, for example, 1.1.1, then t[p] is the (sub) term of t at position p, and t[x/p] denotes t with the substitution of x for the subterm at position p in t. When t' is a term, then t[x/t'] is the usual substition of x for t' in t.

### 6.1 First Algorithm

We consider generalisations w.r.t. > (the empty theory) first. For a given set of terms, the set of all possible generalisations (modulo renaming) is a complete lattice partially ordered w.r.t. > and so finding a minimal generalisation w.r.t. > is always effectively possible. Thus, the intention of presenting the first algorithm is more to explain a particular inductive inference method in some detail and to provide a firm basis for the following propositions than to convince the reader that the problem of generalisation w.r.t. > is recursively solvable. The algorithm consists of two parts: the synthesis of left hand side and the right hand side of the generalisation rule resp. and we assume a list of global variables V.

*Part 1*: This part investigates the left hand sides of the given rules and synthesises the new left hand side. If Part 1 succeeds in a non-trivial way, then the result, which we will denote by l', is $l_1$ (or $l_2$) with substitutions of a number of (not necessarily distinct) variables at the occurrences of the (smallest) differing subterms in each input term $l_1$ and $l_2$. If part 1 succeeds in a trivial way, then the result is a single variable.

*Part 2*: This part tries to establish a link between the replacements in left hand sides of critical pairs/rewrite rules with their corresponding right hand sides and if successful, synthesises the new right hand side.

More precisely, the l.h.s. and r.h.s. of the generalisation rule are given by:

$$
\begin{aligned}
\text{L.H.S}(l_1,l_2) &= l_1\,[x_1/p_1,\dots,x_n/p_n] \\
\text{R.H.S}(l_1,l_2,r_1,r_2) &= r_1\,[x_1/q_{1_1},\dots,x_1/q_{1_{j_1}},\dots,x_n/q_{n_1},\dots,x_1/q_{n_{j_n}}] \\
&\quad \text{if } r_1\,[x_1/q_{1_1},\dots,x_1/q_{1_{j_1}},\dots,x_n/q_{n_1},\dots,x_1/q_{n_{j_n}}] = r_2\,[x_1/q_{1_1},\dots,x_1/q_{1_{j_1}},\dots,x_n/q_{n_1},\dots,x_1/q_{n_{j_n}}] \\
&= \varepsilon \quad \text{otherwise}
\end{aligned}
$$

*where* $p_1, \dots, p_n$ are the positions of the smallest differing subterms of $l_1$ and $l_2$,
$x_1, \dots, x_n$ are variables which do not occur in $l_1$ and $l_2$ and
$(\forall i,j{:}1 \leq i,j \leq n)\ ((l_1[p_i] = l_1[p_j] \wedge l_2[p_i] = l_2[p_j]) \Rightarrow x_i = x_j)$ ,
for each i, $1 \leq i \leq n$,
$s_{i_1}, \dots, s_{i_{k_i}}$ are the positions of $l_1[p_i]$ in $r_1$,
$t_{i_1}, \dots, t_{i_{l_i}}$ are the positions of $l_2[p_i]$ in $r_2$,
$\{q_{i_1}, \dots, q_{i_{j_i}}\} = \{s_{i_1}, \dots, s_{i_{k_i}}\} \cap \{t_{i_1}, \dots, t_{i_{l_i}}\}$ .

If the algorithm succeeds (i.e. the new right hand side is not ε), then the right hand side is denoted by r' and the generated hypothesis is the rewrite rule $l' \rightarrow r'$.

As an example, consider the the first two rules in the first sequence of critical pairs:

**(C1)** f(h(S(0),g(f(x)))) → f(h(S(S(0)),x))
**(C2)** f(h(S(S(0)),g(h(y,x)))) → f(h(S(S(S(0))),x))

The smallest distinct terms in $l_1$ (the l.h.s. of **C1**)) and $l_2$ (the l.h.s. of C3)) occur at position 1.1.1.1: the subterms 0 and S(0) in $l_1$ and $l_2$ resp. They are replaced by a variable y in part 1 and l' = f(h(S(y),g(f(x)))). In part 2 we search for occurrences of 0 and S(0) in $r_1$ and $r_2$ resp. In both cases, the position is exactly 1.1.1.1.1 and the result of replacing the resp. subterms by y is r' = f(h(S(S(y)), x)). The terms 0 and S(0) do not occur elsewhere in $l_1$ or $l_2$; therefore, the generated rewrite rule is

**(G1)** f(h(S(y), g(f(x)))) → f(h(S(S(y)), x)).

We should note that when generating a right hand side, it is very important to consider, for each introduced variable, only the intersection of the positions of the generalised sub-terms in each given term. For example, consider the rules F(0) => f(0,0) and F(succ(0)) => f(succ(0),0). From the first part, we have the generalisation F(x), with x generalising 0 in the first example and succ(0) in the second. Consider now the right hand sides. In the right hand side of the first example, we should apply the generalistion only to the first 0 (position 1.1) to get f(x,0). The position 1.2, the position of the second 0, is not a position of succ(0) in the second example; if we generalised this occurrence of 0 then we would not have the same generalisation for both examples. In the right hand side of the second example, we apply the generalisation to position 1.1, the only occurrence of succ(0) to get f(x,0).

**Lemma**

If the first algorithm processes a finite number of rules $l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n$ and outputs the rewrite rule $l \rightarrow r$, then $l \rightarrow r > \{l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n\}$ and $l \rightarrow r$ is an acceptable, minimal generalisation.

**Proof**

The proof of generalisation obvious; it just uses the substitutions introduced in Part 1. Clearly any rule thus generated is acceptable because the variables used in Part 2 are a subset of the variables used in Part 1. We prove that $l \rightarrow r$ is minimal by contradiction. Assume that $l \rightarrow r$ is not minimal: there exists $l' \rightarrow r' > \{l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n\}$ such that $l \rightarrow r > l' \rightarrow r'$. Because the generalisations possible on the r.h.s. of rules are always a subset of the generalisations on the l.h.s., we need only consider the l.h.s Thus, there must exist a variable v occurring in l, at position p, and a substitution $\sigma(v)=f(x_1,\ldots,t_n)$ such that $l\sigma = l'$. v can only occur in l when there exists two example terms, $l_i$ and $l_j$, say, and substitutions $\sigma_i$ and $\sigma_j$, say, such that $l_i[p] = l\sigma_i[p] = f1(t_1,\ldots,t_n)$, $l_j[p] = l\sigma_j[p] = f2(s_1,\ldots,s_m)$, and $l_j[p] \neq l_i[p]$ . More specifically, $f1 \neq f2$. Because l' is also a generalisation of $l_i$ and $l_j$, there must exist substitutions $\rho_i$ and $\rho_j$, say, such that $l'\rho_i = l_j$, $l\rho_j = l_j$, and therefore $l_i[p] = l'\rho_i[p] = f1(_1,\ldots,t_n)$,$l_j[p] = l'\rho_j[p] = f2(s_1,\ldots,s_m)$. Since l generalises l', then $l'\rho_i[p] = l\sigma\rho_i[p] = f(x_1,\ldots,t_n)\rho_i = f1(_1,\ldots,t_n)$ and $l'\rho_j[p] = l\sigma\rho_j[p] = f(x_1,\ldots,t_n)\rho_j = f2(s_1,\ldots,s_m)$. But there cannot exist an f such that $f = f1$, $f = f2$ and $f1 \neq f2$. Therefore, there cannot exist such a substitution $\sigma$, and thus $l \rightarrow r$ is minimal.

**Corollary**

If the first algorithm processes an infinite number of rules $l_1 \rightarrow r_1$, $l_2 \rightarrow r_2$, ... and outputs the rules $l \rightarrow r$, then $l \rightarrow r > \{ l_1 \rightarrow r_1, l_2 \rightarrow r_2, \ldots \}$ and $l \rightarrow r$ is an acceptable, minimal generalisation.

An inductive inference algorithm is considered as solving a particular learning problem successfully if it converges when applied to a sequence of examples; i.e. after a finite number of steps it generates a hypothesis which will not be changed later when processing more information. This clearly reflects the concept of learning from incomplete information. The natural question is whether or not the algorithm presented is able to generalise all sequences of critical pairs which are generalisable.

**Lemma**

If for some infinite sequence of rules $\{ l_1 \rightarrow r_1, l_2 \rightarrow r_2, \ldots \}$ there exists an acceptable, minimal generalisation w.r.t. >, then the first algorithm solves the learning problem successfully.

*Proof*: (sketch of proof by contradiction).
Assume that there exists an acceptable generalisation $l \rightarrow r$ with $l \rightarrow r > \{l_1 \rightarrow r_1, l_2 \rightarrow r_2, ...\}$, but that the algorithm does not solve the problem. One can easily exclude the case that the algorithm does not terminate for some finite set $\{l_1 \rightarrow r_1, l_2 \rightarrow r_2, ...\}$. An instance of l, perhaps l itself, can always be found by part 1 of the algorithm. Similarly, part 2 can not fail. It remains to determine whether or not the algorithm may have an infinite number of mind changes. Assume that the first algorithm, when processing the infinite sequence $\{l_1 \rightarrow r_1, l_2 \rightarrow r_2, ...\}$ generates an infinite sequence of hypotheses $\{l'_1 \rightarrow r'_1, l'_2 \rightarrow r'_2, ...\}$ where every hypothesis $l'_m \rightarrow r'_m$ is a generalisation of some finite initial segment. For every $l'_i$, $i=1,2,...$, there must be a substitution $\sigma_i$ with $l'_i \sigma_i = l_1$, as all hypotheses are generalisations covering at least the first example. Therefore, the term $l'_i$ cannot be syntactically larger than $l_1$. This means that at least the l.h.s. of $l'_1 \rightarrow r'_1, l'_2 \rightarrow r'_2, ...$ must become stable because there are (up to renaming of variables) only finitely many terms which may be constructed by part 1, and part 1 always computes some term. It generates a new version of l only when the former version becames wrong. As it must reject terms which do not generalise some example, it always keeps a correct generalisation and such a term exists by assumption. We denote this term by $l'$. Therefore, almost all $l'_i$ are equal to $l'$. Now, consider part 2 of the algorithm; similar arguments apply: there must be some term $r'$ which equals almost every $r'_i$. Thus, the construction of the generalisation contradicts the assumption of infinitely many distinct hypotheses.

We can now apply these results to the example investigated above. In particular, we use the knowledge about the underlying term rewriting system which allows us to prove that the distinct subterms occuring in (C1), (C3) , ... form the complete sequence of normal forms 0, S(0), S(S(0)) , ... for the sort **nat** .

Corollary
The first algorithm solves the inductive inference problem given by (C1) , (C3) , ... successfully and generates a generalisation w.r.t. > which is additionally an exact normal generalisation.

As a consequence of this corollary, (G1) is a correct generalisation; i.e. **R1** $\cup$ (G1) is a conservative extension of **R1**.

Now, consider the first two examples in the second sequence of critical pairs (C2), (C4), ... Part 1 of the algorithm proceeds as described above and the resulting term is l' = f(h(S(z), g(h(y, x))). However, in this example, part 2 can not succeed because the terms 0 and S(0) do not occur as subterms of the corresponding right hand sides and after (trivially) applying the generalisation of 0 and S(0) to z, the resulting terms differ. Thus, the algorithm fails. Obviously, the problem is the inability to recognise the similarity between 0 and S(0) on the one hand and y and S(y) on the other hand. The only way to generalise the right hand sides w.r.t. > involves the introduction of a completely new variable which does not occur in any left hand side. However, although this would yield a generalisation w.r.t. >, it would not be an acceptable generalisation.

We now present a second *semi*-algorithm which is an extension of the first and solves this problem by looking for generalisations modulo the theory $\to^*_R$ .

## 6.2 Second (Semi)-Algorithm

*Part 1*: This part is identical to the corresponding part of the first inductive inference algorithm and yields a term l'.

*Part 2*: First, the same procedure as Part2 of the first algorithm is applied in as many positions as possible. Thus, every right hand side may be modified by introducing some variables as before. If the first algorithm terminates with failure, then the second algorithm proceeds to search for a right hand side by searching for a *normal* generalisation as follows. There are still some occurrences in which the modified right hand sides are different, these occurrences are denoted by $v_1, \dots, v_m$. If the corresponding subterm of the right hand side at occurrence $v_j$ is denoted by $t[v_j]$,then we look for the first $w_j$ in the lexicographic ordering of terms from $T_\Sigma(X)$, where X is the set of variables occurring in the right hand sides and any variables introduced into the modified left hand sides, such that for each i, $w_j\sigma_i \to^*_R\ t[v_j]$. We note that the first algorithm is just the special case $w_j s_i\ =\ t[v_j]$. Thus, we have

$$\begin{array}{ll} \text{L.H.S}(l_1,l_2) & = l_1 [x_1/p_1,\dots,x_n/p_n] \\ \text{R.H.S}(l_1,l_2,r_1,r_2) & = r_1 [x_1/q_{1_1},\dots,x_1/q_{1_{j_1}}, \dots, x_n/q_{n_1},\dots,x_1/q_{n_{j_n}}] [w_1/v_1,\dots,w_m/v_m] \end{array}$$

*where*

$p_1, \dots, p_n$ are the positions of the smallest differing subterms of $l_1$ and $l_2$,

$x_1, \dots, x_n$ are variables which do not occur in $l_1$ and $l_2$ and

$$(\forall i,j{:}1 \leq i,j \leq n)\ ((l_1[p_i] = l_1[p_j]\ \wedge\ l_2[p_i] = l_2[p_j]) \Rightarrow x_i = x_j)\ ,$$

for each n, $1 \leq i \leq n$,

$s_{i_1}, \dots, s_{i_{k_i}}$ are the positions of $l_1[p_i]$ in $r_1$,

$t_{i_1}, \dots, t_{i_{l_i}}$ are the positions of $l_2[p_i]$ in $r_2$,

$$\{q_{i_1}, \dots, q_{i_{j_i}}\} = \{s_{i_1}, \dots, s_{i_{k_i}}\} \cap \{t_{i_1}, \dots, t_{i_{l_i}}\}$$

$v_1, \dots, v_m$ are positions of the (smallest) differing subterms in

$r_1 [x_1/q_{1_1},\dots,x_1/q_{1_{j_1}}, \dots, x_n/q_{n_1},\dots,x_1/q_{n_{j_n}}]$ and $r_2 [x_1/q_{1_1},\dots,x_1/q_{1_{j_1}}, \dots, x_n/q_{n_1},\dots,x_1/q_{n_{j_n}}]$,

for each $v_i$, $1 \leq i \leq m$

$w_i$ is the *least* term in $T_\Sigma(\text{var}(r_1) \cup \text{var}(r_2) \cup \{x_1, \dots, x_n\})$

(assuming a total ordering on variables and lexicographic ordering on terms)

such that

$$w_i[x_1/r_1[q_{1_1}], \ldots, x_n/r_1[q_{n_1}]] \to^*_R r_1[v_i]$$
$$w_i[x_1/r_2[q_{1_1}], \ldots, x_n/r_2[q_{n_1}]] \to^*_R r_2[v_i]$$

If the(semi)-algorithm terminates, then the right hand side is denoted by r' and the generated hypothesis is the rewrite rule $l' \to r'$.

As an example, consider the first pair in the first sequence of critical pairs: (C1) and (C3).

| | | | |
|---|---|---|---|
| (C2) | f(h(S(0),g(h(y,x)))) | $\to$ | f(h(S(S(y)),x)) |
| (C4) | f(h(S(S(0)),g(h(y,x)))) | $\to$ | f(h(S(S(S(y))),x)) |

Part 1 results in the introduction of one variable for generalisation, z say, (for terms 0 and S(0) resp.) and the resulting l.h.s. is f(h(S(z),g(h(y,x))). In Part 2, we search for the least r' in $T_\Sigma(\{x,y,z\})$ such that r'[0/z] can be rewritten to f(h(S(S(y)),x)) and r'[s(0)/z] can be rewritten to f(h(S(S(S(y))),x)). Clearly, the first solution for r' is f(h(S(S(z + y)),x)). Thus the generated rewrite rule is

(G2) f(h(S(z),g(h(y,x)))) $\to$ f(h(S(S(z+y)),x))

**Lemma**

If the second algorithm processes a finite number of rules $l_1 \to r_1, \ldots, l_n \to r_n$ and outputs a rewrite rule $l \to r$, then $l \to r > \{l_1 \to r_1, \ldots, l_n \to r_n\}$ and $l \to r$ is an acceptable, minimal generalisation.

**Corollary**

If the second algorithm processes an infinite number of rules $l_1 \to r_1$, $l_2 \to r_2$, ... and outputs a rewrite rule $l \to r$, then $l \to r > \{l_1 \to r_1, \ldots, \} (\to^*_R)$ $l \to r$ is an acceptable, minimal generalisation.

**Lemma**

If for some infinite sequence of rules $\{l_1 \to r_1, l_2 \to r_2, \ldots\}$ there exists an acceptable minimal generalisation w.r.t. $> (\to^*_R)$, then the second algorithm solves the learning problem successfully.

**Corollary**

The second algorithm solves the inductive inference problem given by (C2), (C4), (C6), ... successfully and generates a generalisation w.r.t. $> (\to^*_R)$ which is an exact normal generalisation.

As a consequence of this corollary, (G2) is a correct generalisation; i.e. R1 $\cup$ G2 is a conservative extension of R1.

## 7. CONCLUSIONS

In this paper we have introduced two concepts of generalisation. The first refers to the language of terms and although it is more appropriate to our problem of finding confluent, ground consistent enrichments of non-confluent rewriting systems, we have found that the simpler (usual) syntactic notion may be used in many examples. Two basic inductive inference algorithms for finding generalisations of infinite sequences of terms (or rules) have been presented and shown to be successful for the example under consideration.

Often, a set of terms is not generalisable because we cannot quantify over operators: we have only first order rewriting. Two alternatives to higher-order rewriting, or meta-rewriting (as described in [Kirchener]) are i) to embed the given problem in a richer theory (i.e. add auxiliary sorts, operators and rules) where we can "count" operators and ii) introduce a richer type structure with sub-type relations. Both approaches are currently under investigation and further work is planned.

## ACKNOWLEDGEMENTS

We thank Alfons Geser and Brian Matthews for their helpful comments on this topic and the second author gratefully acknowledges grants from the Nuffield Foundation and Leipzig University of Technology.

## REFERENCES

**[Angluin/Smith]**
D. Angluin, C.H. Smith, A survey of inductive inference: theory and methods, *Computing Surveys* 15 (1983) 3, pp. 237-269.

**[Barzdin]**
J.M. Barzdin, Some rules of inductive inference and their use for program synthesis, *Proc. IFIP 9th World Congress*,Paris, 1983, R.E.A. Mason (ed.), North-Holland, 1983, pp. 333-336.

**[Ehrig/Mahr]**
H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification*1, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985

**[Huet/Oppen]**
G.Huet, D. Oppen, Equations and Rewrite Rules: A Survey, in *Formal Lanaguage Theory: Perspectives and Open Problems*, R. Book (ed.) pp. 349-405, Academic Press, New York, 1980.

**[Kirchener]**
H. Kirchner, Schematization of infinite sets of rewrite rules. Application to the divergence of completion processes, in Proc. Rewriting Techniques and Applications, P. Lescanne (ed.), *Lecture Notes in Computer Science* 256, Springer-Verlag, 1987, pp. 180-191.

**[Kodratoff]**
Y. Kodratoff, Generalizing and particularising as the techniques of learning, *Proc. 2nd Int. Symp. on Artif. Intell. and Inf. Control Systems of Robots*, Smolenice, Czech., I. Plander (ed.), North-Holland,1982, pp. 131-134.

**[Knuth/Bendix]**
D. E. Knuth, P. B. Bendix, Simple Word Problems in Universal Algebras, *Computational Algebra*, J. Leach (ed.), Pergammon Press 1970.

**[Lange]**
St. Lange, A decidability problem of Church-Rosser specifications for program synthesis, Proc. Analogical and Inductive Inference, Wendisch-Rietz, GDR, K.P. Jantke (ed.), *Lecture Notes in Computer Science* 265, Springer-Verlag, 1987, pp.105-124.

**[Plotkin]**
G.D Plotkin, Automatic Methods of Inductive Inference, *Ph.D. Thesis*, Edinburgh University, 1971.