Research
Report
RR-92-22

Unifying Cycles

Jörg Würtz

March 1992

Unifying Cycles

Jörg Würtz

DFKI-RR-92-22

A short version of this report has appeared in the Proceedings of the 1992 European Conference on Artificial Intelligence, Vienna, Austria, August 3-7, 1992.

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITWM-9105).

Unifying Cycles

Jörg Würtz

Abstract

Two-literal clauses of the form $L \leftarrow R$ occur quite frequently in logic programs, deductive databases, and—disguised as an equation—in term rewriting systems. These clauses define a cycle if the atoms L and R are weakly unifiable, i.e., if L unifies with a new variant of R. The obvious problem with cycles is to control the number of iterations through the cycle. In this paper we consider the cycle unification problem of unifying two literals G and F modulo a cycle. We review the state of the art of cycle unification and give new results for a special type of cycles called unifying cycles, i.e., cycles $L \leftarrow R$ for which there exists a substitution σ such that $\sigma L = \sigma R$. Altogether, these results show how the deductive process can be efficiently controlled for special classes of cycles without losing completeness.

Contents

1 Introduction

It is the foremost goal of the research in the field of automated deduction to develop general *and* adequate proof methods and techniques for the logics under consideration. It is comparatively easy to invent a general proof method, but it is much more difficult to develop a general *and* adequate proof technique. For example, the resolution principle [Rob65] and the connection method [Bib87] are general proof methods for first-order logic. But are they adequate? What is the meaning of adequateness in the first place? Roughly speaking, we will consider a technique as being adequate if it solves simpler problems faster than more difficult ones. We illustrate the notion of adequateness by a problem, where the known general proof techniques face difficulties whereas trained humans seem to be able to solve it quite reasonably.

For this purpose, consider the following set of clauses in Prolog-like notation which is taken from [Pfe88] and was originally studied by Lucasiewicz.

$$\begin{array}{llll} & \leftarrow & Pi(iab, i(ibc, iac)). & \text{G} \\ Pw & \leftarrow & Pv,\ Pivw. & \text{MP} \\ Pi(i(ixy, z), i(izx, iux)). & & & \text{A} \end{array}$$

The terms represent implicational formulas, i.e., iab encodes $a \rightarrow b$ and P asserts the derivability of its argument. Thus, the second clause represents modus ponens. It contains several *cycles* [Bib88] defined by the connections between the atom Pw and the atoms $Pivw$ and Pv. The clause MP can be applied to itself and this may lead to an exponential growth of the search space. The obvious problem is to control the self-applicability of MP while retaining completeness. Lukasiewicz has found a 29 step proof. He must have exercised a good control over MP! Quintus Prolog on a Sun SPARC station 2 did not find a proof in several days. Nearly all existing automatic theorem provers cannot solve this problem as well since they are not able to exercise a good control over MP. E. Lusk reports[1] that the parallel version of Otter at Argonne is able to obtain a hyperresolution proof with about 150 proof-steps while generating 6.5 million clauses in about half an hour during the search for it. Their prover does not have a good control over MP as well. It solves the problem by sheer power.

In [BHW91] it was conjectured that a problem like the Lucasiewicz-formula could be solved in less than a second by way of a technique called

[1]Private communication with W. Bibel [BHW91]

cycle unification. At present this conjecture remains a challenge since the Lucasiewicz-formula is a particularly difficult instance of a class of formulas which could eventually be treated by cycle unification. In [BHW91] a first step was made towards this goal by restricting the attention to the special case of formulas with exactly one cycle. In fact, we have focused our analysis on the simple class of two-literal clauses of the form $Pl_1 \dots l_n \leftarrow Pr_1 \dots r_n$ which consists of nothing but a single cycle. This additional restriction simplifies the discussion without loss of generality of the method. In this paper we further analyze clauses consisting of a single cycle and extend the results found in [BHW91].

Such a two-literal clause is usually embedded in the context of some larger formula, or set of clauses. Again for simplicity of the discussion and without loss of generality, we restrict the treatment to the case of two additional clauses, namely a goal clause – referred to as *(calling) goal* – of the form $\leftarrow Ps_1 \dots s_n$, which calls the cycle, and a fact – called *(terminating) fact* – of the form $Pt_1 \dots t_n$, which terminates the cycle. In our restricted case a *cycle unification problem* is then the following one:

> Is there a substitution σ such that $\sigma Ps_1 \dots s_n$ is a logical consequence of $Pl_1 \dots l_n \leftarrow Pr_1 \dots r_n$ and $Pt_1 \dots t_n$?

If such a substitution σ exists, then σ is said to be a *solution* for the cycle unification problem. For more general cases, cycle unification can be defined in an analogue way.

In order to be able to control a cycle we have to answer the following questions. Is cycle unification decidable? How many independent most general solutions has a cycle unification problem? Does there exist a unification algorithm which enumerates a minimal and complete set of solutions for a cycle unification problem? Answers to these questions may help to increase the power of automated theorem provers significantly. For example, if a cycle is embedded in a larger formula and it can be determined that the corresponding cycle unification problem is unsolvable, then the clauses defining the cycle can be eliminated from the formula. If a minimal and complete set Σ of solutions for a cycle unification problem exists and can be enumerated, then any other solution is subsumed by a solution in Σ and need not to be considered. If Σ is finite, then this may prune a potentially infinite search space to a finite one. But theorem proving is not the only task which may benefit from cycle unification.

There are a variety of applications for cycle unification. Observe that although the variables occurring in the two atoms $Ps_1 \dots s_n$ and $Pt_1 \dots t_n$

might not be instantiated, it is possible to analyze the structure of the cycle. Therefore, we can compute partial solutions in a preprocessing manner (all possible solutions if finitely or only a subset otherwise). If some variables will be instantiated in the course of further computation, we can update the partial solutions (see also eg. [Fut88] on partial evaluation). Furthermore, cycle unification helps us to transform recursive programs to iterative ones. The iterative structure can be compiled such that a proof might be detected faster than with the depth-first search of PROLOG. One of the important applications is datalogic, i.e., the field between logic and databases (cf. [Bib87]). It has been shown by eg. Smith [SGG86] that cycles are the source of non-terminating queries. Consequently, insights from cycle unification may be and have already been used to determine non-terminating queries to deductive systems ([DVB89],[DVB90]). Cycle unification might also contribute to answer the question whether the top-down, Prolog-like evaluation of recursive calls can be guaranteed to terminate ([UvG88],[Plü90]).

Although cycle unification is of significant importance for the field of automated deduction, it has received surprisingly little attention in the literature. Function-free cycle unification problems, i.e., cycle unification problems defined over variables and constants only, occur mainly in deductive databases and it can be shown that under certain conditions these problems do not give rise to infinite computations (cf. [MN83]). In [OW84] the number of iterations through a cycle can be limited via a user-defined parameter. In [Vie87] certain cycle unification problems are solved by generalization and subsumption. There, after several iterations through a cycle, subterms occurring in a goal are replaced by variables. Subsumption techniques may now be applied to terminate otherwise infinite derivations. The technique is shown to be complete. Unfortunately, answers to the generalized goal need not to be answers to the initial goal. M. Schmidt–Schauß [SS88] has shown that cycle unification is decidable provided that the goal and the fact are ground, i.e., they do not contain variable occurrences. Independently, P. Devienne [Dev90] has given a more general result for cycle unification problems with linear goals and facts, i.e., each variable occurs at most once in the goal and the fact. He uses essentially the same ideas as Schmidt–Schauß, but a very special technique based on directed weighted graphs. Devienne's results were used by De Schreye et al. [DVB90] to decide whether cycles admit non-terminating queries to deductive systems. Another approach has been taken by H.J. Ohlbach [Ohl90a] who represented sets of terms by so-called abstraction trees which may compress the search space. Moreover, abstraction trees can be used to compile two-literal clauses and in certain cases a finite abstraction tree can represent infinitely many solutions of a cycle unification problem [Ohl90b]. A further approach for unifying infinite sets of terms which are encoded in so called ρ-terms is described in [Sal92]. The incorporation of ρ-terms into logic programming allows on the one hand infinite queries and the finite representation of infinitely many answers. On the other hand, it avoids repeated computation and certain kinds of infinite loops, without changing the denotational semantics of the programs.

In [BHW91] we developed the theoretical foundations for cycle unification. For various classes of restricted cycle unification problems we showed their decidability, proved that they have at most finitely many most general solutions and constructed an algorithm to compute this set. The most general result concerned the class of non-recursive matching cycles $\mathcal{C}_{nrm}$, i.e., cycles $\{L \leftarrow R\}$ for which there exists a substitution σ such that $\sigma L = R$ or $L = \sigma R$ and $\not\exists i : x \mapsto t \in \sigma^i$, $x \in \mathcal{V}ar(t)$, and $t \neq x$.[2] But these were only fundamental classes of cycle unification problems. One of the open problems was to control cycles which overcome the limits of matching cycles. In this paper we present the class of *unifying cycles*, i.e., cycles $\{L \leftarrow R\}$ such that L and R are unifiable. With this class we finish work on cycles whose problems might be characterized by having always finitely many solutions.

After some preliminaries on definitions and notations we formally define cycle unification in Section 3. The important notion of dependency graphs is introduced in Section 4. In Section 5 we define different classes of restricted cycle unification problems and show that their unification problems are decidable, determine the unification type, and develop a unification algorithm. Our most general result concerns the class of unifying cycles $\{L \leftarrow R\}$ which is a combination of the first three analyzed types of cycle unification problems. The paper concludes with a summary of the results on cycle unification and an outline of future work.

2 Definitions and Notations

Our definitions and notations follow those suggested in [DJ91]. Throughout this paper capital letters such as P, Q, ... denote *predicate symbols*, small letters such as a, b, ... denote *constants*, f, g, ... *function symbols*, and

[2]By σ^i we denote the i-fold composition of σ with itself, i.e., $\sigma^1 = \sigma$ and $\sigma^i = \sigma(\sigma^{i-1})$.

z, y, ... *variables.* A *term* is either a variable or of the form $f(t_1, \ldots, t_n)$, where t_1, ..., t_n are terms. s, t, ... denote terms. An *atom* is of the form $P(t_1, \ldots, t_n)$. Let X be an atom or a term. $\mathcal{V}ar(X)$ denotes the set of variables occurring in X. X is called *ground* iff X does not contain any variable. X is called *linear* iff every variable occurs at most once in X. By X^k we denote the syntactic object where each variable occurring in X has the index k attached to it. $t[x_1, \ldots, x_n]$ denotes a term t such that $\{x_1, \ldots, x_n\} \subseteq \mathcal{V}ar(t)$.

A *substitution* is a mapping from the set of variables into the set of terms which is equal to the identity almost everywhere. Hence, it can be represented as a finite set of pairs $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, $x_i \neq t_i$, $1 \leq i \leq n$. Substitutions are denoted by small greek letters such as σ, θ, ... The identity substitution is called ε. $\sigma t = \sigma(t)$ if t is a variable and $\sigma t = f(\sigma t_1, \ldots, \sigma t_n)$ if $t = f(t_1, \ldots, t_n)$. $\mathcal{D}om(\sigma) = \{x \mid x$ is a variable and $\sigma x \neq x\}$ is the *domain* of σ.

The *composition* $\sigma\tau$ of two substitutions σ and τ is defined by $(\sigma\tau)x = \sigma(\tau x)$. The *restriction* of the substitution σ to the set V of variables is defined by $\sigma|_V\, x = \sigma x$ if $x \in V$ and $\sigma|_V\, x = x$ otherwise. A substitution σ is called *variable-pure* if $\{\sigma x \mid x \in \mathcal{D}om(\sigma)\}$ only consists of variables. A *renaming* is a variable-pure substitution σ such that $\sigma x = \sigma y$ implies $x = y$ for $x, y \in \mathcal{D}om(\sigma)$.

If W is a set of variables, then $\sigma = \tau\ [W]$ iff $\forall x \in W : \sigma x = \tau x$. A substitution σ is called *more general* than a substitution τ on W, $\sigma \stackrel{\bullet}{\leq} \tau\ [W]$, iff there exists a substitution ρ such that $\rho\sigma = \tau\ [W]$. Two substitutions σ and τ are called *equivalent* (or *variants*) on W, $\sigma \sim \tau\ [W]$, iff $\sigma \stackrel{\bullet}{\leq} \tau\ [W]$ and $\tau \stackrel{\bullet}{\leq} \sigma\ [W]$. Two substitutions σ and τ are called *independent* on W iff $\sigma \stackrel{\bullet}{\not\leq} \tau\ [W]$ and $\tau \stackrel{\bullet}{\not\leq} \sigma\ [W]$.

σ is called a *unifier* for t and t' iff $\mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$ and $\sigma t = \sigma t'$. A unifier σ of t and t' is called *most general unifier* iff $\sigma \stackrel{\bullet}{\leq} \tau\ [\mathcal{V}ar(t) \cup \mathcal{V}ar(t')]$ for all unifiers τ of t and t'. The definitions above can be extended to atoms, equations, and sets of equations in the obvious way. For a unification algorithm we use the operations suggested in [MM82].

3 Cycle Unification

$C = \{L \leftarrow R\}$ is called a *cyclic theory*, or *cycle* for short, if the atoms L and R are weakly unifiable, i.e., there exist two substitutions σ and σ' such that $\sigma L = \sigma' R$ [Ede85]. Let G and F be two atoms such that $\mathcal{V}ar(G) \cap$

$\mathcal{V}ar(F) = \emptyset$. A *cycle unification problem* $\langle G \xrightarrow[C]{\circ} F\rangle$ (or $\langle G \xrightarrow{\circ} F\rangle_C$) is the problem whether there exists a substitution σ such that σG is a logical consequence of F and C. A substitution σ is a *solution* for the cycle unification problem if $\mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(G)$ and σG is a logical consequence of F and C.[3]

Since solutions to cycle unification problems are substitutions, the notions of more general, independent, etc. substitutions can be extended to more general, independent, etc. solutions of cycle unification problems in the obvious way.

As a first example consider the problem

$$\langle Pa \xrightarrow{\circ} Pfffa\rangle_{\{Px \leftarrow Pfx\}}.$$

The empty substitution ε is the only most general solution for this problem. However, there may be more than one solution as the example

$$\langle Pxy \xrightarrow{\circ} Pab\rangle_{\{Pvw \leftarrow Pwv\}}$$

shows. This problem has the two independent most general solutions $\{x \mapsto a,\ y \mapsto b\}$ and $\{x \mapsto b,\ y \mapsto a\}$. But, there may be even infinitely many independent most general solutions. As an example consider

$$\langle Px \xrightarrow{\circ} Pa\rangle_{\{Pfy \leftarrow Py\}}.$$

This problem has the most general solutions $\{x \mapsto a\}$, $\{x \mapsto fa\}$, $\{x \mapsto ffa\}$, ...

For a cycle unification problem $\langle G \xrightarrow{\circ} F\rangle_{\{L \leftarrow R\}}$ to be solvable, the atoms F and G must be of the form $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$, respectively. Since L and R are weakly unifiable, their predicate symbols must also be identical, i.e., L and R must be of the form $P'(l_1, \ldots, l_n)$ and $P'(r_1, \ldots, r_n)$, respectively. In the sequel we will only consider cycle unification problems of this form. Furthermore, as the case $P \neq P'$ is trivial, we assume $P = P'$.

[3]A cycle unification problem should not be confused with a theory unification problem $\langle G =_C F\rangle$, i.e., the problem whether there exists a substitution σ such that $\sigma G =_C \sigma F$ [Bib87, Sti85].

To solve a cycle unification problem $\langle G \xrightarrow[C]{\circ} F\rangle$ we have to find a substitution which either unifies G and F or unifies – viz. simultaneously unifies each equation in –

$$\mathcal{C}^k = \mathcal{N} \cup \mathcal{Y}^k \cup \mathcal{X}^k,$$

where

$\mathcal{N} = \{s_1 \doteq l_1^1,\ \ldots,\ s_n \doteq l_n^1\}$ is the set of *e$\underline{n}$try equations*,

$\mathcal{Y}^k = \{r_1^i \doteq l_1^{i+1},\ \ldots,\ r_n^i \doteq l_n^{i+1} \mid 1 \leq i \leq k\}$
is the set of *c$\underline{y}$cle equations* for k iterations through the cycle, and

$\mathcal{X}^k = \{r_1^{k+1} \doteq t_1,\ \ldots,\ r_n^{k+1} \doteq t_n\}$
is the set of *e$\underline{x}$it equations* after k iterations through the cycle.

The following proposition is an immediate consequence of the completeness and soundness of the connection method [Bib87] or SLD-resolution, eg. [Llo84].

Proposition 1 *σ is a solution for $\langle G \xrightarrow[C]{\circ} F\rangle$ iff there exists a substitution θ such that θ unifies G and F and $\sigma = \theta|_{\mathcal{V}ar(G)}$ or there exists a natural number k such that θ unifies $\mathcal{C}^k$ and $\sigma = \theta|_{\mathcal{V}ar(G)}$.*

Throughout the paper τ will denote the most general unifier of G and F restricted to $\mathcal{V}ar(G)$, if it exists. Similarly, τ_k will denote the most general unifier of $\mathcal{C}^k$ restricted to $\mathcal{V}ar(G)$, if it exists. The solutions τ_k will be computed by applying the Martelli&Montanari operations [MM82] to the set $\mathcal{C}^k$.[4]

Let $\mathcal{C} = \langle G \xrightarrow[C]{\circ} F\rangle$ be a cycle unification problem. A set Σ of substitutions is a *complete* set of solutions for $\mathcal{C}$ iff each substitution in Σ is a solution for $\mathcal{C}$ and for each solution θ for $\mathcal{C}$ we find a substitution σ in Σ such that $\sigma \lesssim \theta[\mathcal{V}ar(G)]$. A complete set Σ of solutions for $\mathcal{C}$ is said to be *minimal* iff for all $\sigma,\ \theta \in \Sigma$ we find that $\sigma \lesssim \theta[\mathcal{V}ar(G)]$ implies $\sigma = \theta$.

In order to be able to control a cycle, we are interested in the answer to three basic questions. Is cycle unification decidable? How many independent

[4]We start with a set of term-equations. The following operations (in the sequel called Martelli&Montanari operations) are exhaustively applied. Let x be a variable and t a term. If $x \doteq x$ occurs, it is erased. $t \doteq x$ is replaced by $x \doteq t$. If $x \doteq t$ occurs in the set, $x \mapsto t$ is applied to all other equations (variable elimination); if $x \in \mathcal{V}ar(t)$, then failure. If the set contains $f(t_1 \ldots t_n) \doteq f'(t'_1 \ldots t'_n)$ and $f \neq f'$ then failure, otherwise replace this equation by $t_1 \doteq t'_1,\ \ldots,\ t_n \doteq t'_n$ (term reduction). A set of term-equations is in solved form if all equations are of the form $x \doteq t$. We stop with success if we obtain a solved form.

most general solutions has a cycle unification problem? Does there exist a unification algorithm which enumerates a minimal and complete set of solutions for a cycle unification problem?

Following [Sie90], we define the type of a cycle unification problem as follows. A cycle unification problem is of type *unitary* iff there exists a single most general solution, *finitary* iff there exist finitely many most general solutions, and *infinitary* iff there exist infinitely many most general solutions.

4 Dependency Graphs

In this section we introduce the notion of *dependency graphs* for cycle unification problems. All results in this paper depend on certain kinds of paths in the dependency graphs.

Consider the cycle unification problem $C = \langle G \multimap\!\!\to F\rangle_{\{L \leftarrow R\}}$. A dependency graph relates variables occurring in the Goal G and variables of the cycle $\{L \leftarrow R\}$. First, the variables in G are related to the variables occurring in the first instance of the left-hand side of the cycle, i.e., L^1. Second, the variables in the i-th instance of the right-hand side of the cycle, i.e., R^i, are related to the variables in the $i+1$-st instance of the left-hand side, i.e., L^{i+1}. Let $\mathcal{N}'$ and $\mathcal{Y}'^i$ be the sets which are obtained from $\mathcal{N}$ and $R^i \doteq L^{i+1}$ by applying all possible Martelli&Montanari operations, respectively, i.e., they are in solved form. The variable dependency graph $\mathcal{G}_C$ corresponding to C is the pair (V, E), where

- V is a set of nodes, containing a node labeled by v for each $v \in \mathcal{V}ar(C)$, and
- E is a set of directed edges $x \rightsquigarrow y \in E$ computed as follows.
  1. Let $u,\ v \in \mathcal{V}ar(G)$ and $x,\ y \in \mathcal{V}ar(L \leftarrow R)$. If $t[u] \doteq t'[x^1] \in \mathcal{N}'$ or $t'[x^1] \doteq t[u] \in \mathcal{N}'$, then add the directed edge from u to x, i.e., $u \rightsquigarrow x$. If $t[u] \doteq t'[v] \in \mathcal{N}'$, then add the undirected edge between u and v. If $t[x^1] \doteq t'[y^1] \in \mathcal{N}'$, then add the undirected edge between x and y. After all edges are inserted, add the directed edge $u \rightsquigarrow x$ if x is reachable from u via directed (traverseable in both directions) or undirected edges. Finally, delete all undirected edges.
  2. Consider equations in $\mathcal{Y}'^i$. The following cases are considered from top to bottom.

- If $x^i \doteq t[y^{i+1}] \in \mathcal{Y}'^i$ and $z^{i+1} \doteq t'[y^{i+1}] \in \mathcal{Y}'^i$, then add the directed edges $x \rightsquigarrow y$ and $x \rightsquigarrow z$.
- If $x^i \doteq t[y^{i+1}] \in \mathcal{Y}'^i$ and $z^i \doteq t'[y^{i+1}] \in \mathcal{Y}'^i$, then add the directed edges $x \rightsquigarrow y$ and $z \rightsquigarrow y$.
- If $x^i \doteq t[y^{i+1}] \in \mathcal{Y}'^i$, then add the directed edge $x \rightsquigarrow y$.
- If $x^{i+1} \doteq t[y^i] \in \mathcal{Y}'^i$ and $z^{i+1} \doteq t'[y^i] \in \mathcal{Y}'^i$, then add the directed edges $y \rightsquigarrow x$ and $y \rightsquigarrow z$.
- If $x^{i+1} \doteq t[y^i] \in \mathcal{Y}'^i$ and $z^i \doteq t'[y^i] \in \mathcal{Y}'^i$, then add the directed edges $y \rightsquigarrow x$ and $z \rightsquigarrow x$.
- If $x^{i+1} \doteq t[y^i] \in \mathcal{Y}'^i$ then add the directed edge $y \rightsquigarrow x$.

Since $\mathcal{Y}'^i$ is in solved form, no other cases are possible involving different superscripts.

Observe that no superscribed variables occur in the dependency graph. As an example consider the cycle unification problem

$$C = \langle Pu_1u_2u_3u_4u_5 \xrightarrow{\circ} Pababa\rangle_{\{Pyvywz \leftarrow Pxyvyw\}}.$$

We obtain as the most general unifier of $Pu_1u_2u_3u_4u_5$ and $Py^1v^1y^1w^1z^1$ the substitution

$$\sigma_0 = \{u_1 \mapsto y^1,\ u_2 \mapsto v^1,\ u_3 \mapsto y^1,\ u_4 \mapsto w^1,\ u_5 \mapsto z^1\}$$

because

$$\mathcal{N}' = \{u_1 \doteq y^1,\ u_2 \doteq v^1,\ u_3 \doteq y^1,\ u_4 \doteq w^1,\ u_5 \doteq z^1\}.$$

In the first step we obtain Figure 1.
Furthermore, we obtain as the most general unifier of $Px^iy^iv^iy^iw^i$ and $Py^{i+1}v^{i+1}y^{i+1}w^{i+1}z^{i+1}$ the substitution

$$\sigma_i = \{x^i \mapsto y^{i+1},\ y^i \mapsto w^{i+1},\ v^{i+1} \mapsto w^{i+1},\ v^i \mapsto y^{i+1},\ w^i \mapsto z^{i+1}\}$$

because

$$\mathcal{Y}'^i = \{x^i \doteq y^{i+1},\ y^i \doteq w^{i+1},\ v^{i+1} \doteq w^{i+1},\ v^i \doteq y^{i+1},\ w^i \doteq z^{i+1}\}$$

The dependency graph $\mathcal{G}_C$ is finished by the second step. The result is shown in Figure 2.

$$\begin{tikzcd}
 & u_1 \arrow[d] & u_2 \arrow[d] & \\
 & \mathrm{y} & v & \\
u_3 \arrow[ru] & & & \\
u_4 \arrow[r] & w & z & u_5 \arrow[l]
\end{tikzcd}$$

Figure 1: First step of the construction of the dependency graph

$$\begin{tikzcd}
 & u_1 \arrow[d] & u_2 \arrow[d] & \\
x \arrow[r] & y \arrow[r] \arrow[dd] & v \arrow[l, bend left] & \\
u_3 \arrow[ru] & & & \\
u_4 \arrow[r] & w \arrow[r] & z & u_5 \arrow[l]
\end{tikzcd}$$

Figure 2: Final dependency graph

A variable $u \in \mathcal{V}ar(G)$ *depends* on x^i, $x \in \mathcal{V}ar(L \leftarrow R)$, if

$$\{t[u] \doteq t_1[x_1^1] \text{ or } (t[u] \doteq t'_{1'}[x_{1'}^1],\ t''_{1'}[x_{1'}^1] \doteq t_1[x_1^1])\} \cup$$

$$\{t'_j[x_j^j] \doteq t_{j+1}[x_{j+1}^{j+1}] \text{ or } (t'_j[x_j^j] \doteq t'_{j'}[x_{j'}^{j+1}],\ t''_{j'}[x_{j'}^{j+1}] \doteq t_{j+1}[x_{j+1}^{j+1}]) \mid 1 \leq j < i\},\ x_i = x$$

can be derived from $\mathcal{N}$ and $\mathcal{Y}^{i-1}$ by application of Martelli&Montanari operations. Because of the definition there exists a directed path, i.e., a sequence of adjacent directed edges from u to x, of length i traversed in the right direction. Similarly, $y^j \in \mathcal{V}ar(L^j \leftarrow R^j)$ depends on $x^{j+i} \in \mathcal{V}ar(L^{j+i} \leftarrow R^{j+i})$ if for $x_i = x$

$$\{t[y^j] \doteq t_1[x_1^{j+1}] \text{ or } (t[y^j] \doteq t'_{1'}[x_{1'}^{j+1}],\ t''_{1'}[x_{1'}^{j+1}] \doteq t_1[x_1^{j+1}])\} \cup$$

$$\{t'_k[x_k^{j+k}] \doteq t_{k+1}[x_{k+1}^{j+k+1}] \text{ or } (t'_k[x_k^{j+k}] \doteq t'_{k'}[x_{k'}^{j+k+1}],\ t''_{k'}[x_{k'}^{j+k+1}] \doteq t_{k+1}[x_{k+1}^{j+k+1}]) \mid 1 \leq k < i\},$$

i.e., there exists a directed path of length i between y^i and x^{j+i}. Hence, we have a correspondence between the paths of the dependency graph and the dependencies established by the entry- and cycle-equations. We can also say that a variable $u \in \mathcal{V}ar(G)$ *depends* on x^i if an instantiation of x^i may influence via a set of equations in some $\mathcal{C}^k$ the instantiation of u.

A *path* is a list of variables $\langle x_1, \ldots, x_n \rangle$, $n \geq 1$, $\{x_1, \ldots, x_n\} \subseteq \mathcal{V}ar(L \leftarrow R)$, such that $x_1 \rightsquigarrow x_2, \ldots, x_{n-1} \rightsquigarrow x_n$ is a path in the dependency graph and either no edge starts from x_n or x_n is the first variable in $\langle x_1, \ldots, x_n \rangle$ occurring twice. Furthermore, the first variable of a path must be connected with a variable $u \in \mathcal{V}ar(G)$ via one directed edge. A path $\langle x_1, \ldots, x_l \rangle$ is called *linear* iff $x_i \neq x_j$, $1 \leq i, j \leq l$, $i \neq j$. A path $\rho = \langle x_1, \ldots, x_l \rangle$ *contains* the subpath $\tau = \langle y_1, \ldots, y_m \rangle$ iff there exists an $i \geq 1$ and an $j \leq l$ such that $\tau = \langle x_i, \ldots, x_j \rangle$. A path $\tau = \langle x_1, \ldots, x_l \rangle$ is called a *(cyclic) permutation* iff $\langle x_1, \ldots, x_{l-1} \rangle$ is linear and $x_1 = x_l$ A path $\langle x_1, \ldots, x_l \rangle$ is called a *(cyclic) permutation with linear entry-path* iff $\langle x_1, \ldots, x_{l-1} \rangle$ is linear and there exists an j, $1 < j < l$, such that $x_l = x_j$. It is obvious that a permutation with linear entry-path $\langle x_1, \ldots, x_l, \ldots, x_{l+k}, x_l \rangle$ can be divided into the linear entry-path $\langle x_1, \ldots, x_l \rangle$ and the permutation $\langle x_l, \ldots, x_{l+k}, x_l \rangle$. x is called a *branching-point* if more than one directed edges start from x. Finally, a path $\langle x_1, \ldots, x_l \rangle$ is of *length* l, x_1 is called the *starting-point* and x_l is called the *end-point* of the path.

The cycle unification problem

$$\langle Pu_1u_2u_3u_4u_5 \xrightarrow{\circ} Pababa \rangle_{\{Pyvywz \leftarrow Pxyvyw\}}$$

depicted in Figure 2 defines the permutation

$$\langle y, v, y\rangle$$

and the linear paths

$$\langle v, y, w, z\rangle,\ \langle y, w, z\rangle,\ \langle w, z\rangle,\ \text{and}\ \langle z\rangle.$$

One should observe that the dependency graph contains a cycle iff $\{L \leftarrow R\}$ defines a permutation. Furthermore, for a path $\langle x_1, \dots, x_n\rangle$ one should observe that the superscribed variable x_1^i depends on the superscribed variable x_n^{i+n-1}.

A similar approach with so called argument/variable graphs was undertaken by J. Naughton [Nau89] and with connection graphs by Wei et al. [WLH91]. Both works were settled in the field of deductive databases. They considered a set of facts and introduced non-recursive predicates, i.e., others than the cycle predicates. They disallowed, however, multiple occurrences of variables in the cycle and needed not to consider function-symbols. Thus, these approaches are too weak for cycle unification.

5 Unifying Cycles ($\mathcal{C}_u$)

In this section we show for certain kinds of subclasses of $\mathcal{C}_u$ their decidability, determine the unification type and develop a unification algorithm. These classes show characteristics which can be generalized for the class $\mathcal{C}_u$ of unifying cycles.

A variable x is called *recursive* iff it is possible to derive from $L \doteq R$ with Martelli&Montanari operations an equation of the form $x \doteq t$ such that $x \in \mathcal{V}ar(t)$ and $t \neq x$. Therefore, if the Martelli&Montanari algorithm runs into an occur-check failure, we know that the cycle contains a recursive variable. As an example consider the cycle $\{Pfx \leftarrow Px\}$. Obviously, the variable x is recursive. Embedded in the cycle unfication problem $\langle Py \multimap\!\!\rightarrow Pa\rangle_{\{Pfx \leftarrow Px\}}$ we obtain the independent most general solutions $\{y \mapsto a\}$, $\{y \mapsto fa\}$, $\{y \mapsto ffa\}$, ..., i.e., infinitely many. This is one of the reasons why we exclude recursive variables. A cycle unification problem $\langle G \multimap\!\!\rightarrow F\rangle_{\{L \leftarrow R\}}$ is called *unifying* iff L and R are unifiable, i.e., the cycle contains no recursive variables.

5.1 Linear Paths ($\mathcal{C}_{lp}$)

We recall that a path $\langle x_1, \ldots, x_l \rangle$ is called linear iff $x_i \neq x_j$, $1 \leq i, j \leq l$, $i \neq j$. A cycle unification problem $\langle G \multimap\!\!\rightarrow F \rangle_{\{L \leftarrow R\}}$ is in the class $\mathcal{C}_{lp}$, if the corresponding dependency graph defines only linear paths. Let $\langle x_{1,i}, \ldots, x_{l_i,i} \rangle$, $1 \leq i \leq n$, be the n defined linear paths. Furthermore, let $m = \max(l_1, \ldots, l_n)$ where max denotes the maximum-predicate.

Proposition 2 *Let $\langle G \multimap\!\!\rightarrow F \rangle_{\{L \leftarrow R\}} \in \mathcal{C}_{lp}$ and m be defined as above. Then, for the most general solutions*

$$\tau_{m-1} = \tau_i, \ \ i \geq m,$$

holds.

Proof: After $m-1$ iterations through the cycle every variable occurring in G either does not depend on a variable at all or it depends on a variable $x^j_{l_i,i}$, $1 \leq i \leq n$, $1 \leq j \leq m$. Because the variables $x_{l_i,i}$, $1 \leq i \leq n$, are end-points of linear paths, they do not depend on any other variable. Hence, further iterations through the cycle do not contribute to further solutions. qed

Proposition 2 states that only the first $m-1$ iterations through the cycle contribute to a possible solution of a cycle unification problem defining linear paths. We conclude that for cycle unification problems defining linear paths and m defined as above, we have only to consider τ, i.e., the restriction of the most general unifier of G and F to $\mathcal{V}ar(G)$, if it exists, and the first $m-1$ iterations through the cycle to obtain all possible most general solutions for a cycle unification problem in the class $\mathcal{C}_{lp}$. Conversely, if neither G and F are unifiable nor any of the sets $\mathcal{C}^i$, $0 \leq i < m$, is solvable, then the cycle unification problem is unsolvable.

As an example consider the cycle unification problem

$$\langle Pu_1u_2u_3u_4 \multimap\!\!\rightarrow Paafb{,}fc \rangle_{\{Pfy,zvw \leftarrow Pxxfy,v\}}.$$

The corresponding dependency graph is depicted in Figure 3.
We obtain the linear paths

$$\langle y, v, w \rangle, \ \langle z \rangle, \ \langle v, w \rangle, \ \langle w \rangle$$

such that $m = \max(3, \ 1, \ 2, \ 1) = 3$. If we solve

$$\mathcal{C}^0 = \{ Pu_1u_2u_3u_4 \doteq Pfy^1, z^1v^1w^1, \ Px^1x^1fy^1, v^1 \doteq Paafb{,}fc \},$$

Figure 3:

$$\begin{tikzcd}
& x \arrow[r] \arrow[d] & y \arrow[r] & v \arrow[r] & w \\
u_2 \arrow[r] & z & u_1 \arrow[u] & u_3 \arrow[u] & u_4 \arrow[u]
\end{tikzcd}$$

we obtain the solution

$$\tau_0 = \{u_1 \mapsto fb,\ u_3 \mapsto fc\}.$$

Solving

$$\mathcal{C}^1 = \left\{ \begin{array}{c} Pu_1u_2u_3u_4 \doteq Pfy^1, z^1v^1w^1,\ Px^1x^1fy^1, v^1 \doteq Pfy^2, z^2v^2w^2, \\ Px^2x^2fy^2, v^2 \doteq Paafb,fc \end{array} \right\}$$

yields

$$\tau_1 = \{u_1 \mapsto fc\}.$$

If we iterate once more through the cycle, we obtain from solving

$$\mathcal{C}^2 = \left\{ \begin{array}{c} Pu_1u_2u_3u_4 \doteq Pfy^1, z^1v^1w^1,\ Px^1x^1fy^1, v^1 \doteq Pfy^2, z^2v^2w^2, \\ Px^2x^2fy^2, v^2 \doteq Pfy^3, z^3v^3w^3,\ Px^3x^3fy^3, v^3 \doteq Paafb,fc \end{array} \right\}$$

the solution

$$\tau_2 = \{u_1 \mapsto fy^1\}.$$

Because y^1 depends on w^3, which does not occur in the right-hand side of the cycle $\{Pfy,zvw \leftarrow Pxxfy,v\}$, no further iterations through the cycle contribute new solutions, i.e., $\tau_2 = \tau_j,\ j > 2$.

5.2 Permutations ($\mathcal{C}_p$)

We recall that a path $\langle x_1, \ldots, x_l \rangle$ is a permutation iff $\langle x_1, \ldots, x_{l-1} \rangle$ is linear and $x_1 = x_l$. A cycle unification problem $\langle G \xrightarrow{\circ} F \rangle_{\{L \leftarrow R\}}$ is in the class $\mathcal{C}_p$ if the corresponding dependency graph defines only permutations. Let $\langle x_{1,i}, \ldots, x_{k_i,i}, x_{1,i} \rangle$, $1 \leq i \leq n$, be the n defined permutations. Furthermore, let $N = \text{lcm}(k_1,\ \ldots,\ k_n)$ where lcm denotes the least common multiple.[5]

[5]Because the permutations $\langle x_1, \ldots, x_l \rangle$, $\langle x_2, \ldots, x_1 \rangle$, $\ldots$, $\langle x_l, \ldots, x_{l-1} \rangle$ are defined by the same path in the dependency graph, we will not distinguish between them.

Proposition 3 *Let* $\langle G \multimap F\rangle_{\{L\leftarrow R\}} \in \mathcal{C}_p$ *and* N *be defined as above. Then,*

$$\tau_j \lesssim \tau_{j+i\cdot N},\ j \geq 0,\ i > 0.$$

Proof: Assume τ_j and $\tau_{j+i\cdot N}$ to exist. First, we associate with each variable x occurring in a permutation precisely one permutation such that $\langle x\rangle$ is a subpath of it. We assume $\langle x\rangle$ to be contained in the permutation $\langle x_1,\ldots,x_p,x_1\rangle$. Hence, x^{j+1} depends on x^{j+1+p}, $x^{j+1+2\cdot p}$, ... Because $\forall j\exists l : l\cdot k_j = N$, x^{j+1} depends also on $x^{j+1+i\cdot N}$ and N is by definition the least number for which this is true for all variables.
For computing a solution τ_j we have to solve $\mathcal{C}^j = \mathcal{N}\cup\mathcal{Y}^j\cup\mathcal{X}^j$. A subpath $\langle x,y\rangle$ of the permutation $\langle x_1,\ldots,x_p,x_1\rangle$ is defined by the following sets of equations:

$$\{x^i \doteq y^{i+1}\} \text{ or } \{y^{i+1} \doteq x^i\} \text{ or } \{x^i \doteq t,\ y^{i+1} \doteq t\}$$

such that the last set is equivalent to $\{x^i \doteq y^{i+1},\ x^i \doteq t\}$. This holds because the cycle is non-recursive. The sets above can be derived from the cycle-equations $\mathcal{Y}^i$. Therefore, we obtain the chain of equations

$$x^{j+1} \doteq y_1^{j+2},\ y_1^{j+2} \doteq y_2^{j+3},\ \ldots,\ y_{i\cdot N-1}^{j+i\cdot N} \doteq x^{j+1+i\cdot N}$$

if we follow the permutation-path associated with x. So, x^{j+1} is not only depending on $x^{j+1+i\cdot N}$ but they must be equal, i.e.,

$$x^{j+1} \doteq x^{j+1+i\cdot N}$$

which is yielded by $i\cdot N-1$ variable-elimination steps.
Let $u \in \mathcal{V}ar(G)$. u depends on some variables z_1^{j+1}, ..., z_n^{j+1} such that $x \in \{z_1^{j+1},\ \ldots,\ z_n^{j+1}\}$. For all z_1^{j+1}, ..., z_n^{j+1} the following holds.
Assume $s|_v = f(\ldots)$ for an occurrence[6] v where $u\mapsto s \in \tau_j$. Observe that $f(\ldots)$ represents a constant if the arity of f is 0. From $x^{j+1} \doteq x^{j+1+i\cdot N}$ (which is equivalent to $x^{j+1+i\cdot N} \doteq x^{j+1}$) and from $t'[x^{j+1+i\cdot N}] \doteq t$ (which is obtained from the set of equations $\{R^{j+1+i\cdot N} \doteq F\}$) it follows by variable-elimination that

$$t'[x^{j+1}] \doteq t$$

is implied by $\mathcal{C}^{j+i\cdot N}$. On the other hand, we obtain $t'[x^{j+1}] \doteq t$ from $\{R^{j+1} \doteq F\}$ of $\mathcal{C}^j$ as well. Hence, we also obtain $s|_v = f(\ldots)$ if we

[6]An occurrence is a list of natural numbers or λ. Let t be a term. $t|_\lambda = t$. If $t = f(t_1,\ldots,t_n)$ and v is an occurrence in t_i, then $t|_{i.v} = t_i|_v$.

compute the solution $\tau_{j+i\cdot N}$ for $\mathcal{C}^{j+i\cdot N}$. Because no further operations which are caused by remaining equations in $\mathcal{C}^{j+i\cdot N}$ can make the function-symbol f dissappear. Thus, the topmost function-symbol of $s|_v$ must be equal for j and $j+i\cdot N$ iterations through the cycle.
On the other hand, assume u not to be instantiated or $s|_v \in \mathcal{V}ar$ for j iterations. Because in $\mathcal{C}^{j+i\cdot N}$ we have more equations than in $\mathcal{C}^j$, the value of u after j iterations must be more general than that for $j+i\cdot N$ iterations if $\tau_{j+i\cdot N}$ exists.
This argumentation holds for each z_i^{j+1}, $1 \leq i \leq n$, such that u depends on it. The solutions are related by $\lessdot$ and not by $=$ because they are variants of each other (observe the variable-chain of the permutation-variables). qed

One should observe that the existence of τ_j does not imply the existence of $\tau_{j+i\cdot N}$, e.g. for intertwined permutations like the second example below. But it is easy to see that the non-existence of τ_j implies the non-existence of $\tau_{j+i\cdot N}$. Proposition 3 expresses that we only have to consider the unifier of G and F restricted to $\mathcal{V}ar(G)$, if it exists, and the first $N-1$ iterations through the cycle to obtain all possible most general solutions for a cycle unification problem in the class of permutations.

As an example for a cycle unification problem in $\mathcal{C}_p$ consider

$$\langle Pu_1u_2u_3u_4 \ \stackrel{\circ}{\longrightarrow}\ Pabab\rangle_{\{Pvxyx \leftarrow Pxvxy\}}$$

which defines the dependency graph of Figure 4.

Figure 4:

$$\begin{tikzcd}
 & & u_2 \arrow[d] & & \\
u_1 \arrow[r] & v \arrow[r] & x \arrow[r] \arrow[l, bend left] & y \arrow[l, bend right] & u_3 \arrow[l] \\
 & & & u_4 \arrow[ul] &
\end{tikzcd}$$

The dependency graph defines the two permutations

$$\langle x, y, x\rangle \text{ and } \langle x, v, x\rangle$$

such that $N = \mathrm{lcm}(2,\ 2) = 2$. Considering one instance of the cycle and solving

$$\mathcal{C}^0 = \{Pu_1u_2u_3u_4 \doteq Pv^1x^1y^1x^1,\ Px^1v^1x^1y^1 \doteq Pabab\}$$

yields

$$\tau_0 = \{u_1 \mapsto b,\ u_2 \mapsto a,\ u_3 \mapsto b,\ u_4 \mapsto a\}.$$

From

$$\mathcal{C}^1 = \{Pu_1u_2u_3u_4 \doteq Pv^1x^1y^1x^1,\ Px^1v^1x^1y^1 \doteq Pv^2x^2y^2x^2,\ Px^2v^2x^2y^2 \doteq Pabab\}$$

we obtain the solution

$$\tau_1 = \{u_1 \mapsto a,\ u_2 \mapsto b,\ u_3 \mapsto a,\ u_4 \mapsto b\}.$$

If we iterate once more, we have to solve

$$\mathcal{C}^2 = \left\{ \begin{array}{c} Pu_1u_2u_3u_4 \doteq Pv^1x^1y^1x^1,\ Px^1v^1x^1y^1 \doteq Pv^2x^2y^2x^2, \\ Px^2v^2x^2y^2 \doteq Pv^3x^3y^3x^3,\ Px^3v^3x^3y^3 \doteq Pabab \end{array} \right\}$$

which results again in τ_0 which also implies that $\tau_0 \lesssim \tau_2$.

A slight variation of the example above is the cycle unification problem

$$\langle Pu_1u_2u_3u_4 \ \multimap\!\!\rightarrow\ Pabac\rangle_{\{Pvxyx \leftarrow Pxvxy\}}$$

where we have replaced the fact $Pabab$ with $Pabac$. If we want to solve

$$\mathcal{C}^1 = \{Pu_1u_2u_3u_4 \doteq Pv^1x^1y^1x^1,\ Px^1v^1x^1y^1 \doteq Pv^2x^2y^2x^2,\ Px^2v^2x^2y^2 \doteq Pabac\},$$

x^1 has to be bound simultaneously to b and c such that neither τ_1 nor τ_2 exists.

As another example consider the cycle unification problem

$$\langle Pu_1u_2u_3u_4u_5 \ \multimap\!\!\rightarrow\ Pv_1v_2v_3fa, v_4\rangle_{\{Pyxvwx \leftarrow Pxyxvw\}}$$

which defines the dependency graph depicted in Figure 5.
We obtain $N = \mathrm{lcm}(2,\ 3) = 6$. Solving

$$\mathcal{C}^0 = \{Pu_1u_2u_3u_4u_5 \doteq Py^1x^1v^1w^1x^1,\ Px^1y^1x^1v^1w^1 \doteq Pv_1v_2v_3fa, v_5\}$$

yields the solution

$$\tau_0 = \{u_3 \mapsto fa\}$$

Figure 5:

$$\begin{tikzcd}
w \arrow[rrrr, bend left, ""] & & v \arrow[ll] & & x \arrow[ll] \arrow[rr] & & y \arrow[ll, bend right, ""] \\
u_4 \arrow[u] & & u_3 \arrow[u] & u_5 \arrow[ur] & & u_2 \arrow[ul] & u_1 \arrow[u]
\end{tikzcd}$$

and solving $\mathcal{C}^5$ yields the solution

$$\tau_5 = \{u_1 \mapsto fa,\ u_2 \mapsto fa,\ u_3 \mapsto fa,\ u_4 \mapsto fa,\ u_5 \mapsto fa\}.$$

This confirms that $\tau_0 \preceq \tau_5$. But we also observe that it is not necessary to consider the fourth and fifth iteration at all. This holds because after already 3 iterations all variables in the goal $Pu_1u_2u_3u_4u_5$ are instantiated with fa. Consider the variable x^1. For 2 and 3 iterations we obtain $x^1 \doteq x^3$ and $x^1 \doteq x^4$, respectively. On the other hand, we obtain for x^2 after 2 iterations $x^2 \doteq x^4$. Hence, the chain $x^1 \doteq x^2$, $x^2 \doteq x^3$, $x^3 \doteq x^4$ holds. Because all variables in the first instance depend on some x^i, $1 \le i \le 4$, only 3 iterations are necessary since all further iterations do not change the solution. Let $L = \{p_1, \ldots, p_n\}$ be the lengths of the defined permutations minus 1. For cycles with $\gcd(p_1, \ldots, p_n)^7 = 1$ and a dependency graph consisting of precisely one connected component we are looking for the least number m such that

$$\forall j,\ 1 \le j \le m, \exists u : (j = \sum_{k=1}^{u} i_k{\cdot}p_{l_k},\ i_k \in \{-1,\ 1\},\ p_{l_k} \in L \wedge \forall v, 1 \le v \le u : 0 \le \sum_{k=1}^{v} i_k{\cdot}p_{l_k} \le m).$$

Thus, we get a refinement of the upper limit of iterations we have to consider.

5.3 Permutations with Linear Entry-Path ($\mathcal{C}_{plp}$)

We recall that a path $\langle x_1, \ldots, x_l\rangle$ is a permutation with linear entry-path iff $\langle x_1, \ldots, x_{l-1}\rangle$ is linear and there exists an j, $1 < j < l$, such that $x_l = x_j$. A cycle unification problem $\langle G \multimap F\rangle_{\{L \leftarrow R\}}$ is in the class $\mathcal{C}_{plp}$ if the corresponding dependency graph defines only paths which are

[7]gcd denotes the greatest common divisor.

permutations with linear entry-path or which are a subpath of a permutation with linear entry-path. Let $\langle x_{1,i}, \ldots, x_{l_i,i}, \ldots, x_{l_i+k_i,i}, x_{l_i,i} \rangle$, $1 \leq i \leq n$, be the n defined permutations with linear entry-path. Furthermore, let $m = \max(l_1, \ldots, l_n)$ and $N = \mathrm{lcm}(k_1, \ldots, k_n)$.

Proposition 4 *Let* $\langle G \xrightarrow{\circ} F \rangle_{\{L \leftarrow R\}} \in \mathcal{C}_{plp}$, m *and* N *be defined as above. Then,*

$$\tau_{m-1+i} \mathrel{\dot{\preceq}} \tau_{m-1+i+k \cdot N}, \;\; i \geq 0, \;\; k > 0.$$

Proof: It is a straightforward conclusion from the structure of the dependency graph that after $m-1$ iterations through the cycle all variables $u \in \mathcal{V}ar(G)$ depend either on no variable at all or on variables x^m which are contained in the permutation-parts of the permutations with linear entry-path. After $m-1+i$ iterations through the cycle all $u \in \mathcal{V}ar(G)$ depend either on no variable at all or on variables y^{m+i} such that y is contained in a permutation-part. Now we apply the same argumentation as in the proof of Proposition 3. qed

Proposition 4 tells us that we only have to consider the unifier of G and F restricted to the variables occurring in G, if it exists, and the first $m+N-2$ iterations through the cycle to obtain all possible most general solutions for a cycle unification problem in the class $\mathcal{C}_{plp}$.

As an example consider the cycle unfication problem

$$\langle Pv_1v_2v_3v_4 \xrightarrow{\circ} Pabfc, u \rangle_{\{Pxyfz, fy \leftarrow Pwxfy, fz\}}.$$

The problem defines the dependency graph illustrated in Figure 6.

Figure 6:

$$\begin{tikzcd}
 & & u_1 \arrow[d] & u_2 \arrow[dr] & & u_4 \arrow[dl] & & & \\
w \arrow[rr] & & x \arrow[rr] & & y \arrow[rr] & & z \arrow[ll, bend left] & & u_3 \arrow[ll]
\end{tikzcd}$$

The dependency graph defines the permutation with linear entry-path

$$\langle x, y, z, y \rangle.$$

We conclude $N = 2$ from the permutation-part $\langle y, z, y\rangle$ and $m = 2$ from the linear entry-path $\langle x, y\rangle$, which are defined by the cycle $\{Pxyfz,fy \leftarrow Pwxfy,fz\}$. Therefore, we conclude by Proposition 4 that

$$\tau_{1+i} \preceq \tau_{1+i+k\cdot 2},\ i \geq 0,\ k > 0.$$

As an example we compute τ_3 from

$$\mathcal{C}^3 = \left\{ \begin{array}{c} Pv_1v_2v_3v_4 \doteq Px^1y^1fz^1, fy^1,\ Pw^1x^1fy^1, fz^1 \doteq Px^2y^2fz^2, fy^2, \\ Pw^2x^2fy^2, fz^2 \doteq Px^3y^3fz^3, fy^3,\ Pw^3x^3fy^3, fz^3 \doteq Px^4y^4fz^4, fy^4, \\ Pw^4x^4fy^4, fz^4 \doteq Pabfc,u \end{array} \right\}.$$

We obtain

$$\tau_3 = \{v_1 \mapsto c,\ v_3 \mapsto fc,\ v_4 \mapsto fz^4\}.$$

From solving

$$\mathcal{C}^1 = \left\{ \begin{array}{c} Pv_1v_2v_3v_4 \doteq Px^1y^1fz^1, fy^1,\ Pw^1x^1fy^1, fz^1 \doteq Px^2y^2fz^2, fy^2, \\ Pw^2x^2fy^2, fz^2 \doteq Pabfc,u \end{array} \right\}$$

we obtain the solution

$$\tau_1 = \{v_1 \mapsto c,\ v_3 \mapsto fc,\ v_4 \mapsto fz^2\}.$$

Hence, $\tau_1 \preceq \tau_3$ holds.

5.4 Unifying Cycles ($\mathcal{C}_u$)

Unifying cycles consist of a combination of linear paths, permutations and permutations with linear entry-path where each variable can be a starting-point of certain kinds of paths. We assume the unifying cycle to contain p permutations

$$\langle x_{1,i}, \ldots, x_{m_i,i}, x_{1,i}\rangle,\ 1 \leq i \leq p,$$

pl restricted permutations with linear entry-path

$$\langle y_{1,i}, \ldots, y_{l_i,i}, y_{l_i+1,i}, \ldots, y_{l_i+n_i,i}, y_{l_i+1,i}\rangle,\ 1 \leq i \leq pl,$$

such that no subpath is a subpath of a permutation except the permutation-part of the path itself, and l restricted linear paths

$$\langle z_{1,i}, \ldots, z_{\bar{l}_i,i}\rangle,\ 1 \leq i \leq l,$$

such that no subpath is a subpath of a permutation. Let $m = \max(1,\, l_1,\, \ldots,\, l_{pl},\, \tilde{l}_1,\, \ldots,\, \tilde{l}_l)$ and $N = \operatorname{lcm}(1,\, m_1,\, \ldots,\, m_p,\, n_1,\, \ldots,\, n_{pl})$. Herein, 1 is needed if there are no linear paths and no permutations at all, respectively. We use the abbreviations

$$Perm^j = \{x^j_{1,i},\, \ldots,\, x^j_{m_i,i} \mid 1 \le i \le p\}$$

and

$$Lin^j = \{z^j_{\tilde{l}_1,1},\, \ldots,\, z^j_{\tilde{l}_l,l}\}.$$

Perm and *Lin* denote the sets of variables without superscribed indices. Therefore, *Perm* contains only variables which occur in permutations and *Lin* contains only the end-points of the restricted linear paths. Observe that all permutation-parts of permutations with linear entry-path are contained in the set of permutations.

Proposition 5 *Let* $\langle G \xrightarrow{\circ} F\rangle_{\{L\leftarrow R\}} \in \mathcal{C}_u$, m *and* N *be defined as above. Then,*

$$\tau_{m-1+k} \lessdot \tau_{m-1+k+j\cdot N},\ \ k \ge 0,\ \ j > 0.$$

Proof: Due to better readability we only prove $\tau_{m-1} \lessdot \tau_{m-1+j\cdot N}$. The whole proof of Proposition 5 is established with the addition of one more superscribed index.

Assume τ_{m-1} and $\tau_{m-1+j\cdot N}$ to exist. First we associate with each variable $x \in Perm$ precisely one permutation such that $\langle x\rangle$ is a subpath of it. We assume $\langle x\rangle$ to be contained in the associated permutation $\langle y_1, \ldots, y_l, y_1\rangle$. Let $u \in \mathcal{V}ar(G)$.

First, we consider $m-1$ iterations through the cycle. Assume that u depends on a variable $y^j \in Lin^j$, $j \le m$. Hence, all further iterations leave the value of y^j invariant because y depends on no other variable since it is an end-point.

On the other hand, assume that u depends the first time on a variable x^i, $i \le m$, where $x \in Perm$. Because of the definition of m, every path starting in u must end in an end-point $y \in Lin$ or reach such an $x \in Perm$ in the first $m-1$ iterations. Assume x^i to depend on $y^{i+1}_{j_1}$, $1 \le j_1 \le l$. If there is a branching in the path of the permutation (or at x itself), this cannot be caused by a derived equation of the form $z^i \doteq t$ such that t is no variable because of the non-recursiveness of the cycle $\{L\leftarrow R\}$.

For computing a solution τ_j we have to solve $\mathcal{C}^j = \mathcal{N} \cup \mathcal{Y}^j \cup \mathcal{X}^j$. A subpath $\langle x, y\rangle$ of the permutation $\langle x_1, \ldots, x_p, x_1\rangle$ is defined by the following sets of equations in $\mathcal{Y}'^i$:

$$\{x^i \doteq y^{i+1}\} \text{ or } \{y^{i+1} \doteq x^i\} \text{ or } \{x^i \doteq t,\ y^{i+1} \doteq t\}$$

such that the last set is equivalent to $\{x^i \doteq y^{i+1},\ x^i \doteq t\}$. This holds because the cycle is non-recursive. The sets above can be derived from the cycle-equations $\mathcal{Y}^i$. Therefore, we obtain after $m-1$ iterations through the cycle the chain of equations

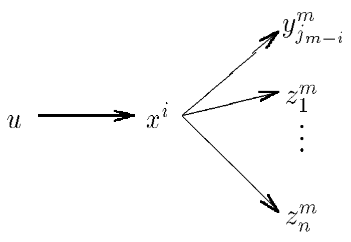
$$x^i \doteq y^{i+1}_{j_1},\ \ldots,\ y^{m-1}_{j_{m-i-1}} \doteq y^m_{j_{m-i}},\ y_k \in \{y_1,\ \ldots,\ y_l\},\ j_1 \le k \le j_{m-i},$$

if we follow the permutation-path associated with x. Hence, it follows from this chain that

$$x^i \doteq y^m_{j_{m-i}}$$

holds. Furthermore, x^i may depend on some other variables $z^m_1,\ \ldots,\ z^m_n$. This is depicted in Figure 7.[8]

Figure 7:



In combination with the set of equations $\{R^m \doteq F\}$ of $\mathcal{C}^{m-1}$, these dependencies establish a set of dependencies for x^i. On the other hand, we know from the proof of Proposition 3 that after $m-1+j\cdot N$ iterations

$$y^m_{m-i} \doteq y^{m+j\cdot N}_{m-i}$$

holds. Since $\langle x\rangle$ is a subpath of $\langle y_1, \ldots, y_l, y_1\rangle$, there must exist a p such that $m-i$ is equal to $m+j\cdot N-p$ (cf. Figure 8). From $m-i = m+j\cdot N-p$ we conclude that $p = j\cdot N+i$ holds. It follows from the underlying chain of equations that

$$x^i \doteq x^{j\cdot N+i}$$

holds. Furthermore, $x^{j\cdot N+i}$ depends on $z^{m+j\cdot N}_1,\ \ldots,\ z^{m+j\cdot N}_n$ where $z_1,\ \ldots,\ z_n$ are the same variables as for $m-1$ iterations. In combination with the set

[8]Here and in Figure 8 the directed edges denote a path of adjacent directed edges in the original dependency graph where the inner nodes are omitted due to better readability.

of equations $\{R^{m+j\cdot N} \doteq F\}$ of $\mathcal{C}^{m-1+j\cdot N}$ results the same set of dependencies for $m-1+j\cdot N$ iterations as for $m-1$ iterations. This is depicted in Figure 8.[9]

Figure 8:

$$\begin{tikzcd}
u \arrow[r] & x^i \arrow[r] & y^m_{j_{m-i}} \arrow[r] & x^{j\cdot N+i} \arrow[r] \arrow[dr] \arrow[dddr] & y^{m+j\cdot N}_{j_{m-i}} \\
 & & & & z_1^{m+j\cdot N} \\
 & & & & \vdots \\
 & & & & z_n^{m+j\cdot N}
\end{tikzcd}$$

This argumentation holds for all x^i such that $\langle x\rangle$ is contained in a permutation. Now we consider again the cases where $s|_v = f(\ldots)$, u is not instantiated or $s|_v \in \mathcal{V}ar$ for $u \mapsto s \in \tau_{m-1}$. The proof is established analogously to the proof of Proposition 3. qed

Proposition 5 tells us that we only have to consider the unifier of G and F restricted to $\mathcal{V}ar(G)$, if it exists, and the first $m+N-2$ iterations through the cycle to obtain all possible most general solutions for a cycle unification problem in the class $\mathcal{C}_u$. Conversely, if neither G and F are unifiable nor any one of the sets $\mathcal{C}^i$, $0 \le i \le m+N-2$, is solvable, then the cycle unification problem is unsolvable. One should observe that this result subsumes the result of linear paths (where $N=1$), of permutations (where $m=1$) and of permutations with linear entry-path. Observe that the existence of τ_{m-1+k} does not imply the existence of $\tau_{m-1+k+j\cdot N}$. But if τ_{m-1+k} does not exist, $\tau_{m-1+k+j\cdot N}$ does not exist as well.

As an example we resume the example of Section 4, i.e., we consider the cycle unification problem

$$\langle Pu_1u_2u_3u_4u_5 \overset{\circ}{\longrightarrow} Pababa\rangle_{\{Pyvywz \leftarrow Pxyvyw\}}.$$

The dependency graph of Figure 9 defines the restricted linear paths

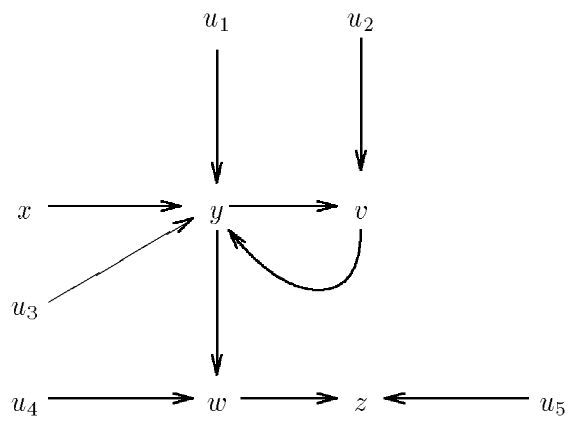
$$\langle w, z\rangle \text{ and } \langle z\rangle$$

[9]Observe that $x^{j\cdot N+i}$ need not to be on the path between $y^m_{j_{m-i}}$ and $y^{m+j\cdot N}_{j_{m-i}}$.

and the permutation

$$\langle y, v, y\rangle.$$

Figure 9:



$$\begin{tikzcd}
 & u_1 \arrow[d] & u_2 \arrow[d] & \\
x \arrow[r] & y \arrow[r] \arrow[d] & v \arrow[l, bend left] & \\
u_3 \arrow[ru] & & & \\
u_4 \arrow[r] & w \arrow[r] & z & u_5 \arrow[l]
\end{tikzcd}$$

Here we see why we must restrict the definition of linear paths. With the former definition we would also have to consider the paths $\langle v, y, w, z\rangle$ and $\langle y, w, z\rangle$. But they contain variables (v and y) which are already subpaths of permutations. Hence, they must not contribute to m. Therefore, we compute $m = \max(1,\ 2,\ 1) = 2$ and $N = \mathrm{lcm}(1,\ 2) = 2$ such that $m + N - 2 = 2$. In order to compute the solution for 1 iteration through the cycle we have to solve

$$\mathcal{C}^1 = \left\{ \begin{array}{c} Pu_1u_2u_3u_4u_5 \doteq Py^1v^1y^1w^1z^1,\ Px^1y^1v^1y^1w^1 \doteq Py^2v^2y^2w^2z^2, \\ Px^2y^2v^2y^2w^2 \doteq Pababa \end{array} \right\}$$

which results in

$$\tau_1 = \{u_1 \mapsto a,\ u_2 \mapsto b,\ u_3 \mapsto a\}.$$

If we solve

$$\mathcal{C}^3 = \left\{ \begin{array}{c} Pu_1u_2u_3u_4u_5 \doteq Py^1v^1y^1w^1z^1,\ Px^1y^1v^1y^1w^1 \doteq Py^2v^2y^2w^2z^2, \\ Px^2y^2v^2y^2w^2 \doteq Py^3v^3y^3w^3z^3,\ Px^3y^3v^3y^3w^3 \doteq Py^4v^4y^4w^4z^4, \\ Px^4y^4v^4y^4w^4 \doteq Pababa \end{array} \right\}$$

we obtain

$$\tau_3 = \{u_1 \mapsto a,\ u_2 \mapsto b,\ u_3 \mapsto a\},$$

i.e., τ_1 again which is implied by Proposition 5.

Let $\langle G \mathrel{-\!\!\circ\!\!\!\rightarrow} F\rangle_{\{L\leftarrow R\}}$ be a cycle unification problem in the class $\mathcal{C}_u$. The following steps define a cycle unification algorithm for unifying cycles with the help of the previous propositions. Algorithms for $\mathcal{C}_{lp}$, $\mathcal{C}_p$, and $\mathcal{C}_{plp}$ are special cases.

Unification Algorithm for $\mathcal{C}_u$

1. If G and F are unifiable, then compute τ as the most general unifier for G and F restricted to the variables in G.

2. Compute the dependency graph for $\langle G \mathrel{-\!\!\circ\!\!\!\rightarrow} F\rangle_{\{L\leftarrow R\}}$.

3. If $\langle G \mathrel{-\!\!\circ\!\!\!\rightarrow} F\rangle_{\{L\leftarrow R\}} \in \mathcal{C}_u$, then compute the lengths $l_1, \ldots, l_i$ of all defined restricted linear paths/linear entry-paths and the lengths $m_1, \ldots, m_j$ of all defined permutations. Let $m = \max(1, l_1, \ldots, l_i)$ and $N = \text{lcm}(1, m_1 - 1, \ldots, m_j - 1)$.

4. If $\mathcal{C}^k$ is solvable, then compute τ_k as the most general unifier for $\mathcal{C}^k$, restricted to the variables occurring in G, $0 \le k \le m+N-2$.

5. Let Σ be the set of solutions obtained in steps (1) and (4). If $\Sigma = \emptyset$, the problem is unsolvable. Otherwise, iteratively eliminate a substitution α if the current set of solutions contains another substitution δ with $\delta \lesssim \alpha\,[\mathcal{V}ar(G)]$. The obtained set is a minimal and complete set of solutions for the cycle unification problem $\langle G \mathrel{-\!\!\circ\!\!\!\rightarrow} F\rangle_{\{L\leftarrow R\}}$.

As an example we resume our example from above. An application of the algorithm yields the following results.

1. $Pu_1u_2u_3u_4u_5$ and *Pababa* are unifiable by $\tau = \{u_1 \mapsto a,\ u_2 \mapsto b,\ u_3 \mapsto a,\ u_4 \mapsto b,\ u_5 \mapsto a\}$.

2. The dependency graph is depicted in Figure 2.

3. The dependency graph defines the restricted linear paths $\langle w, z\rangle$ and $\langle z\rangle$ and the permutation $\langle y, v, y\rangle$. Therefore, $m = 2$ and $N = 2$.

4. $\tau_0 = \{u_1 \mapsto b,\ u_2 \mapsto a,\ u_3 \mapsto b,\ u_4 \mapsto a\}$, $\tau_1 = \{u_1 \mapsto a,\ u_2 \mapsto b,\ u_3 \mapsto a\}$, $\tau_2 = \{u_1 \mapsto b,\ u_2 \mapsto a,\ u_3 \mapsto b\}$ are the most general solutions obtained by solving $\mathcal{C}^0, \mathcal{C}^1$, and $\mathcal{C}^2$, respectively.

5. We obtain the set $\{\tau_1,\ \tau_2\}$ as a minimal and complete set of solutions.

The following theorem follows immediately from the previous results. Observe that Theorem 6 holds for $\mathcal{C}_{lp}$, $\mathcal{C}_p$, and $\mathcal{C}_{plp}$ as well because they are subsets of $\mathcal{C}_u$.

Theorem 6 *Let C be a unifying cycle.*
(i) $\langle G \xrightarrow[C]{\circ} F\rangle$ *is decidable.*

(ii) $\langle G \xrightarrow[C]{\circ} F\rangle$ *is finitary.*

(iii) *There exists an algorithm computing a minimal and complete set of solutions for* $\langle G \xrightarrow[C]{\circ} F\rangle$.

6 Summary and Future Work

In this paper we firstly defined cycle unification. We then restricted our attention to the class $\mathcal{C}_u$ which denotes the class of cycle unification problems defining unifying cycles, i.e., cycles $\{L \leftarrow R\}$ for which L and R are unifiable. By considering several subclasses of $\mathcal{C}_u$, leading in combination to the results for $\mathcal{C}_u$, we have extended known work.

Table 1 gives an overview of our results as well as of previous work. In each row we state the decidability and the unification type for a particular class of cycle unification problems, indicate whether there exists an algorithm to compute a minimal and complete set of solutions and provide the reference if there exists one. $\mathcal{C}$ denotes the class of unrestricted cycle unification problems. In $\mathcal{C}_l$ and $\mathcal{C}_g$ goals and facts are restricted to be linear and ground, respectively. $\mathcal{C}_m$ contains only matching cycles $\{L \leftarrow R\}$ such that there exists a substitution σ and $\sigma L = R$ or $L = \sigma R$. $\mathcal{C}_{nrm}$ contains only non-recursive matching cycles $\{L \leftarrow R\}$, i.e., the cycle is matching and $\nexists i : x \mapsto t \in \sigma^i$, $x \in \mathcal{V}ar(t)$, and $t \neq x$. The various classes are related as shown in Figure 10.

Our most general result concerns the class of unifying cycles. For this class we have shown that we only have to consider finitely many iterations

Class	Decidability	Type	Algorithm	References
$\mathcal{C}$	open	infinitary	open	
$\mathcal{C}_l$	decidable	infinitary	open	[Dev90]
$\mathcal{C}_g$	decidable	unitary	yes	[SS88]
$\mathcal{C}_m$	open	infinitary	open	
$\mathcal{C}_{nrm}$	decidable	finitary	yes	[BHW91]
$\mathcal{C}_u$	decidable	finitary	yes	in this paper

Table 1: Properties of cycle unification classes

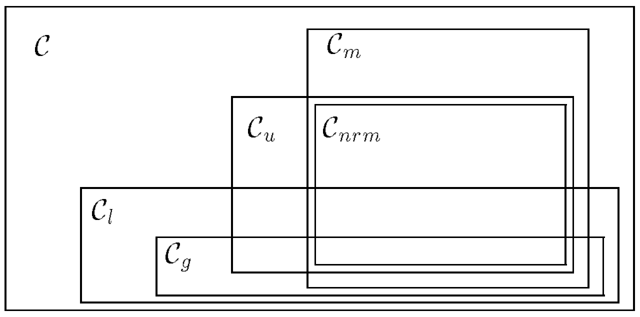


Figure 10: The relation between the classes $\mathcal{C}$, $\mathcal{C}_l$, $\mathcal{C}_g$, $\mathcal{C}_m$, $\mathcal{C}_{nrm}$, and $\mathcal{C}_u$.

through the cycle to obtain a minimal and complete set of solutions. Furthermore, we have presented an algorithm for computing the maximal number of necessary iterations to obtain this set. This enables us to efficiently control the deductive process without losing completeness. Thus, we have finished the basic research for cycle unification problems which are non-recursive. For future work on these classes we are interested in refinements for the upper bound of iterations through the cycle. A first approach has been shown at the end of Section 5.2 for intertwined permutations. Further basic research has to consider the case of recursive cycles, i.e., cycles which can admit infinitely many independent solutions.

One of the major open problems in our restricted context is the question whether $\mathcal{C}$ is decidable. $\mathcal{C}_l$ and $\mathcal{C}_u$ are decidable. However, there are several results which point into the opposite direction for the case of $\mathcal{C}$. In [Dau88] it is shown that the termination of a one rule term rewriting system, where rewriting may occur at proper subterms, is undecidable. Similarly, we know from [SS88] that the class of Horn clauses consisting of two clauses of the form $L \leftarrow R$ and two ground unit clauses is undecidable. It is, however, not obvious, how these results could be adapted to cycle unification problems.

In the future we intend to develop heuristics to control further classes of cycle unification problems. We are looking for a wellfounded ordering based on a measure of complexity for instances of the cycle in order to apply an idea similar to the one contained in [SS88]. Certain cycles $\{L \leftarrow R\}$ cause some of the terms occurring in L and R to grow or shrink monotonically in each iteration of the cycle. If there were an upper bound for these terms defined by G or F, then one would be able to decide the cycle unification problem $\langle G \overset{\circ}{\longrightarrow} F\rangle_{\{L\leftarrow R\}}$. For illustration of this idea consider the cycle unification problem

$$\langle Pffx,x \ \overset{\circ}{\longrightarrow}\ Pufu\rangle_{\{Pfffy,fz\leftarrow Pfy,z\}}.$$

The i-th instance of the right-hand side of the cycle $\{Pfffy,fz \leftarrow Pfy,z\}$ is matched against the $i+1$-st instance of the left-hand side by $\sigma_i = \{y^i \doteq ffy^{i+1},\ z^i \doteq fz^{i+1}\}$. We observe that the depth of y and z decreases with each iteration through the cycle. The goal and the fact define upper bounds because of their non-linearity which correlates y and z. In [BHW91] we have exploited this insight for the computation of the number k of iterations through the cycle to obtain a solution. For the example above we obtain $k = 2$ and the solution $\tau_2 = \{x \mapsto f^5y^3\}$. Under some circumstances, those problems can be solved with a technique called meta-unifying which is described in [Sal92]. We expect other useful heuristics to exist.

Acknowledgement: I would like to thank Wolfgang Bibel, Gerd Große and Steffen Hölldobler at the Technische Hochschule Darmstadt for their valuable comments on this paper. The work was partially supported by the ESPRIT-project MEDLAR and the Stadt Dreieich.

References

[BHW91] W. Bibel, S. Hölldobler, and J. Würtz. Cycle unification. Forschungsbericht AIDA–91–15, TH Darmstadt, August 1991.

[Bib87] W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, Braunschweig, 2 edition, 1987.

[Bib88] W. Bibel. Advanced topics in automated deduction. In R. Nossum, editor, *Fundamentals of AI II.* Springer Verlag, 1988.

[Dau88] M. Dauchet. Termination of rewriting is undecidable in the one–rule case. In *Mathematical Foundations of Computer Science*, pages 262–270. LNCS 324, 1988.

[Dev90] P. Devienne. Weighted graphs: A tool for studying the halting problem and time complexity in term rewriting systems and logic programming. *Journal of Theoretical Computer Science*, 75:157–215, 1990.

[DJ91] N. Dershowitz and J.-P. Jouannaud. Notations for rewriting. In *EATACS Bulletin*, pages 162–172, 1991.

[DVB89] D. De Schreye, K. Verschaetse, and M. Bruynooghe. On the existence of non–terminating queries for a restricted class of Prolog–clauses. *Artificial Intelligence*, 41:237–248, 1989.

[DVB90] D. De Schreye, K. Verschaetse, and M. Bruynooghe. A practical technique for detecting non–terminating queries for a restricted class of horn clauses, using directed, weighted graphs. In *Proceedings of the International Conference on Logic Programming*, pages 649–663, 1990.

[Ede85] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1:31–46, 1985.

[Fut88] Y. Futamura. Program evaluation and generalized partial computation. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 685–692, 1988.

[Llo84] J.W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation – Artificial Intelligence. Springer-Verlag, 2 edition, 1984.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4(2):258–282, 1982.

[MN83] J. Minker and J.M. Nicolas. On recursive axioms in deductive databases. *Information Systems*, 8(1):1–13, 1983.

[Nau89] J.F. Naughton. Data independent recursion in deductive databases. In *Journal of Computer and System Sciences*, pages 259–289, 1989.

[Ohl90a] H.J. Ohlbach. Abstraction tree indexing for terms. In *Proceedings of the European Conference on Artificial Intelligence*, 1990.

[Ohl90b] H.J. Ohlbach. Compilation of recursive two–literal clauses into unification algorithms. In *Proceedings of the AIMSA*, 1990.

[OW84] H.J. Ohlbach and G. Wrightson. Solving a problem in relevance logic with an automated theorem prover. In *Proceedings of the Conference on Automated Deduction*, pages 496–508, 1984.

[Pfe88] F. Pfenning. Single axioms in the implicational propositional calculus. In *Proceedings of the Conference on Automated Deduction*, pages 710–713. Lecture Notes in Computer Science, Springer, 1988.

[Plü90] L. Plümer. *Termination Proofs for Logic Programs*, volume 446 of *Lecture Notes in Computer Science, Springer*. Springer, 1990.

[Rob65] J.A. Robinson. A machine–oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Sal92] G. Salzer. The unification of infinite sets of terms and its applications. Technical report, Technische Universität Wien, 1992.

[SGG86] D.E. Smith, M.R. Genesereth, and M.L. Ginsberg. Controlling recursive inference. *Artificial Intelligence*, 30:343–389, 1986.

[Sie90] J. Siekmann. An introduction to unification theory. In R. B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 369–424, 1990.

[SS88] M. Schmidt-Schauß. Implication of clauses is undecidable. *Journal of Theoretical Computer Science*, 59:287–296, 1988.

[Sti85] M.E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

[UvG88] J.D. Ullman and A. van Gelder. Efficient tests for top–down termination of logical rules. *Journal of the ACM*, 35(2):345–373, 1988.

[Vie87] L. Vieille. Recursive query processing: The power of logic. Technical report, ECRC, 1987.

[WLH91] S.S. Wei, W. Lu, and I.M. Hsu. Using multiple query optimization technique to minimize relation searches in processing bounded recursion. In *International Symposium on Artificial Intelligence*, pages 143–149, 1991.