Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

Solving Equality Reasoning Problems with a Connection Graph Theorem Prover

Axel Präcklein

SEKI Report SR-90-07

Solving Equality Reasoning Problems with a Connection Graph Theorem Prover*

Axel Präcklein
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, D-6750 Kaiserslautern
prckln@informatik.uni-kl.de UUCP

April 18, 1990

Abstract

The integration of a Knuth-Bendix completion algorithm into a paramodulation theorem prover on the basis of a connection graph resolution procedure is presented. The Knuth-Bendix completion idea is compared to a decomposition approach, and some ideas to handle conditional equations are discussed. The contents of this paper is not intended to present new material on term rewriting, instead it is more a pleading for the usage of completion ideas in automated deduction. It records our experience with an actual implementation of a hybrid system, where a completion procedure was imbedded into a connection graph theorem prover, the MKRP-system, with satisfactory positive results.

Keywords: Equality reasoning, Knuth-Bendix algorithm, paramodulation, resolution.

Contents

*This Research was supported by the Deutsche Forschungsgemeinschaft, SFB 314, D2.

1 Introduction

The equality predicate can be described in a logical calculus by specifying the following three axioms and two axiom schemata which are sufficient to define equality in first order logic. It is done in this or a very similar way in all introductory books about logic like these of E. Mendelson [Men87, 3rd edition], H.-D. Ebbinghaus, J. Flum, and W. Thomas [EFT78], and V. Sperschneider [Spe84].

Definition 1.1 (Equality Axioms)

- $\forall x: \ x = x$ *(reflexivity)*
- $\forall x, y: \ x = y \ \Rightarrow \ y = x$ *(symmetry)*
- $\forall x, y, z: \ x = y \wedge y = z \ \Rightarrow \ x = z$ *(transitivity)*
- *For each function symbol f and each argument of this function an axiom:*
  $\forall x_1, \ldots, x_n, y: \ x_i = y \Rightarrow f x_1 \ldots x_i \ldots x_n = f x_1 \ldots y \ldots x_n$
- *For each predicate symbol P and each of its arguments an axiom:*
  $\forall x_1, \ldots, x_n, y: \ P x_1 \ldots x_i \ldots x_n \wedge x_i = y \Rightarrow P x_1 \ldots y \ldots x_n$

Of course it is very inefficient to handle the equality predicate automatically using these axioms. Hence research on the mechanization of the equality predicate has led to a variety of different special calculi. Some of them, for example paramodulation [RW69] and RUE-Resolution [Dig79], integrate a new rule into the existing resolution calculus, others like E-Resolution [Mor69] and the more recent general E-unification approaches [Blä86,YS86] ignore the context of the local equality problem and consider only equations without conditions.

There are some approaches to classify equality reasoning methods. One of them distinguishes between term replacement and difference reduction methods [Blä86]. Term replacement works by substituting terms using equations, difference reduction considers at least two terms and tries to make them equal by inserting equations at top level and revising the subterms recursively for terms with the same function symbol.

The term "difference reduction" is used at two levels of abstraction. Firstly it denotes term decomposition as explained above. This is on the same level as term replacement. Secondly it means that the whole approach is on a higher (AI) level and reduces semantical differences [Blä86].

We propose a further classification according to the axioms and theorems containing the equality predicate but we shall not focus on it in this paper. It concerns the structure of equations with regard to similarity of their left and right hand sides, and with accordance to similarities between whole equations. Commutativity for example is an axiom, which is itself structured in the sense that its left and right hand side are very similar. The single axioms for left and right zero however are examples for formulae without such a structure. Corresponding properties can be attached to theorems. Additionally theorems can be classified according to their relation to axioms. Especially induction theorems have themselves structure and a strong relation to their hypotheses (for example $x(y+z) = xy + xz \Rightarrow (x+1)(y+z) = (x+1)y + (x+1)z$) [Hut89]. Another dimension for the classification is whether there is just one theorem or several theorems. Many theorems can occur when equality is imbedded in a resolution prover via E-Resolution, namely one for each pair of literals with the same predicate and opposite sign.

This classification induces corresponding reasoning methods. Structure in the axioms induces the usage of a decomposition approach, structure in the theorems induces special transformation methods [Hut89]. Many theorems in combination with structured axioms induce a graph based decomposition method to store partial solutions to be shared such that they can be used at different positions. In the unstructured case rewriting should bring the literals to normal form, and then they should be unifiable.

But the most interesting case of equality reasoning is when conditional equations occur. All advanced equality reasoning methods mentioned above are led astray when formulae like $A \Rightarrow x = y$ or $A \Rightarrow x = c$ (see examples 5.2 and 5.4) are among the axioms. Such conditional equations are typical for real situations and neither do they have structure nor are they directable, and there is no reason to believe that the "equality problem" is solved when a satisfactory procedure for handling the unit equations is found.

Nevertheless we also focus our attention on sets of unit equations because these theories are so far better researched and must be the entry point for an efficient and powerful equality prover.

2 Equality Reasoning

Research on the mechanization of the equality predicate can be classified into three areas: unification theory, the development of special deduction rules for equality, and term rewriting systems. We shall briefly sketch the work in these areas.

Unification The simplest case of working with equality is to make two objects equal by the replacement of subobjects by others. Unification is the process to find a uniform replacement for the variables in terms such that these terms become syntactically equal, which means that they can be written as the same string. The endomorphism describing the replacement for the variables is called a substitution. A unifier is a substitution that makes the terms equal. $\{x \mapsto b, y \mapsto a\}$, for example, is a unifier of $f(x, a)$ and $f(b, y)$. In the following we often discard the parentheses of the terms, when the arities of the function symbols are clear.

To come closer to the mathematical equality relation the notion of unification can be extended to E-unification, where a set E of equations is given as axioms, which induce an equivalence relation that is written $=_E$. An example for a unifier of $f(a, x)$ and $f(b, y)$ under the theory E=$\{f(x, y) = f(y, x)\}$ of commutativity is $\{x \mapsto b, y \mapsto a\}$.

In general a unification problem can have more than one solution. J. Robinson [Rob65] proved that in the case with empty E there exists (up to renaming of variables) a unique most general unifier representing the whole set of solutions whenever this set is not empty. In arbitrary theories there is not necessarily such a representative unique unifier. The next step was to extend the concept to sets of unifiers, which fulfill the requirements to be correct, complete, and minimal. But there are theories for which such sets do not exist. Hence equational theories can be classified as to whether for each unifiable set of terms the set of most general unifiers has only one element, is finite, infinite, or does not exist at all [Sie88]. The corresponding theories are called unitary, finitary, infinitary, and nullary.

One task in unification theory is to develop algorithms to compute sets of unifiers. A universal unification algorithm is one working for all theories, usually this notion is also used for algorithms handling whole classes of theories. One main goal in this area is to combine known unification algorithms for special theories to new ones for more complex theories. However there are problems: for example the algorithms for associative unification and commutative unification could not be combined to an algorithm for theories that have both properties, and this is the case for almost all theories. In general a new algorithm must be designed for such combined theories. Currently the most advanced approach is M. Schmidt-Schauß' method for the combination of unification algorithms [SS87]. It works for arbitrary disjunct theories and free function symbols.

Deduction A general purpose deduction system must handle all combinations of equations, even if they occur together with other predicates in the same formula, as for example in $\forall n : \mathit{Even}(n) \Leftrightarrow (\exists m : n = 2m)$.

The handling of equality via the axioms in definition 1.1 is very inefficient and hence J. Darlington [Dar68], E. Siebert [Sib69], J. Robinson [Rob65], and G. Robinson and L. Wos [RW69] incorporated the equality relation into automated deduction systems by designing new inference rules. The best known inference rule is paramodulation, which works on two clauses where one of them contains an equality literal. One side of the equation must be unifiable with a term t in the other clause by a substitution σ. Then the paramodulant consists of all

literals of the two clauses without the equality literal after replacing the term t by the other side of the equation and applying the substitution σ to all literals of the new clause. R. Kowalski showed [Kow75] that using the paramodulation rule enhances the power of deduction systems.

Paramodulation is a deduction rule that is applicable 'almost everywhere' making search graphs very bushy [Bun83], and so it should only be used if the result is of overriding importance for other arguments in the proof.

To transform two literals into resolvable ones using equations is the motivation of E-Resolution [Mor69] and RUE-Resolution [Dig79] and we shall come back to this type of reasoning in section 3.

Rewriting The observation that equations can be 'applied' to terms led to a term replacement approach for the treatment of the equality relation. To obtain an algorithm to prove the equality of two terms one can successively apply equations to the terms. Such an algorithm only decides the equality of the terms but can not make them equal by computing an instantiation of their variables as required for resolution based systems. The main idea is to consider the equations as rules that can only be applied in one direction. The direction is determined by an ordering on the set of terms.

A method to decide the equality of two terms under special equality theories can then be obtained by "reducing" the terms to a unique normal form using the directed equations. The theory axioms must obey certain conditions, they must be confluent and Noetherian, to ensure completeness and termination of the decision procedure. The equations defining the theory must be directable and must have the properties above or it must be possible to add other equations such that the new system is equivalent to the old one and has the desired properties. This procedure developed by D. Knuth and P. Bendix [KB70] is called completion. The new system of directed equations constitutes a set of rewriting rules.

When computing a normal form of a term all situations where two rules can be applied to derive different successors are dangerous because it must be ensured that both cases later on lead to the same normal form. D. Knuth and P. Bendix showed that it is enough to consider critical pairs between rules and to add corresponding equations to ensure this property. Critical pairs can be constructed from two rules or two instances of the same rule if the left hand sides of the rules overlap, that means that some subterm of the left hand side can be unified with the other left hand side. One term of the critical pair is the right hand side of the first rule with the unifier applied to it. For the other term the unifiable subterm in the one left hand side is replaced by the other right hand side and again the unifier is applied to the result.

In principle the Knuth-Bendix completion algorithm then works as follows [KB70,HO80,Buc85,Der87,JL87]: Beginning with a set of undirected equations, an empty set of directed rules, and a reduction ordering it tries to derive a convergent set of rules from the equations. It applies the following steps until no equations remain: Take an equation, apply all rules to the equation, direct the equation according to the given reduction ordering, and put it into the set of rules. Generate all critical pairs, that is, terms for which rule applications overlap, between the new rule and the set of rules and put them into the set of equations. If this algorithm terminates, it produces a set of rules that can be used to decide the equality of arbitrary terms of the given theory.

A rule is applicable to a term if the left hand side of the rule matches the term or a subterm of it. If a rule is applied to an object with subterms to which it is applicable, then these are replaced by the right hand side of the rule with the matcher applied to it. In the field of Automated Deduction the application of the rules is often called demodulation [WRCS67,WOLB84] and we will use this term here too.

Of course there are interrelations between unification theory and term rewriting systems and one goal is to combine rewriting techniques and unification algorithms.

Some results of the research in term rewriting systems led to universal unification algorithms restricted to so called confluent or canonical theories. F. Fages [Fag83], J. Hullot [Hul80], J.-P. Jouannaud, C. Kirchner, H. Kirchner [JKK83], J. You, P. Subramayou [YS86], A. Martelli, C. Moiso, G. Rossi [MMR86], and C. Kirchner [Kir85] defined systems for this purpose.

Sometimes special theory unification algorithms are used in completion systems. Such a method was used for example by M. Stickel [Sti85] to prove ring commutativity from $x^3 = x$. We shall recourse to this point in section 5.

There are also interrelations between special deduction rules and rewriting systems; we shall come back to them in section 5 too.

3 Decomposition

Decomposition was first used by J. Herbrand in his thesis [Her30]. With this concept we mean the method to derive unifiers for the subterms of the given terms and to combine these solutions to solve the equality problem for the whole terms. A. Martelli and U. Montanari [MM82] exploited this 'divide and conquer' strategy for an alternative to the unification algorithm of J. Robinson [Rob65]. The kernel of the unification algorithm based on decomposition is given in definition 3.1.

Definition 3.1 (Unification by Transformation Rules)
A unification problem is a set of equations. It is in solved form when each equation has the form $x = t$ with x not occurring anywhere else in the equation set. The following rules are performed on a set of equations until no rule is applicable. If the system is in solved form in this final situation the derived set of equations represents a solution, else no solution exists.

1. *Switching: Replace an equation $t = x$ by $x = t$.*
2. *Deletion: Delete $t = t$.*
3. *Decomposition: Replace $fs_1 \ldots s_n = ft_1 \ldots t_n$ by $s_1 = t_1, \ldots, s_n = t_n$.*
4. *Elimination: replace all occurrences of x by t in all other equations if $x = t$ is an equation where x does not occur in t.*

A. Martelli and U. Montanari refined this version using special datastructures and labelings and obtained an almost linear unification algorithm. Of course they used another representation of unifiers because the exponentiality of Robinson-unification stems from the term replacement property of idempotent unifiers.

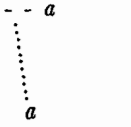
One advantage of the usage of nondeterministic rules is that the order of operations is easier to control and unessential conditions need not be checked in the control mechanism. This can make correctness and completeness proofs for theory unification algorithms much easier.

C. Kirchner [Kir85] invented a conceptual framework to include special equality theories in such a rule based algorithm. J. Gallier [GS86,GS89] and K. Bläsius [Blä86] concurrently described universal unification algorithms via rules. J. Gallier used a Martelli-Montanari-like version for the pure unification part, whereas K. Bläsius unifies with a Robinson procedure. In addition K. Bläsius' approach is more implementation oriented and proposes special graph structures for storing the information about the unification state.

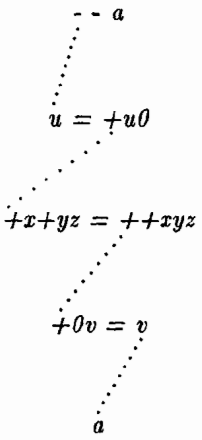
The basis for our work was the system of K. Bläsius and so we implemented an improved version of his rule system, where the Robinson rules are replaced by Martelli-Montanari-rules because these fit better into the general framework.

We demonstrate the usage of the rules with the help of a classical example, namely that $- - x = x$ in a group. Unsolved subproblems are indicated by dashed lines, solved subproblems by complete lines labeled with unifiers. Equality chains are written in the text $term_1 - l_1 = r_1 - \cdots - l_n = r_n - term_2$, two terms concatenated with — must always have the same function symbol or must be variables, no equation is allowed to occur more than once in one chain.

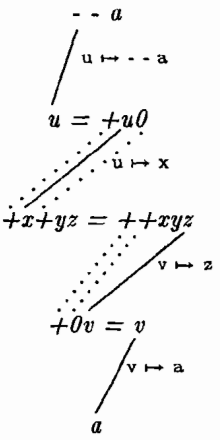
Example 3.2 (Group, Involution)



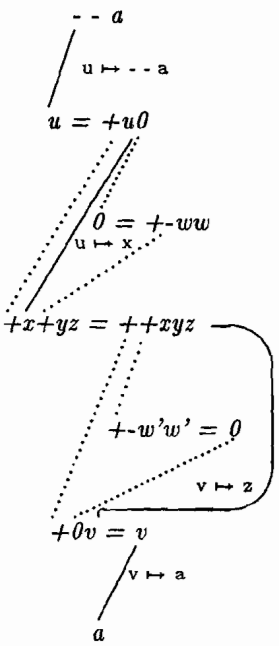
is the initial termgraph, that is, - - a and a are to be made equal. The dashed line indicates the problem to be solved.



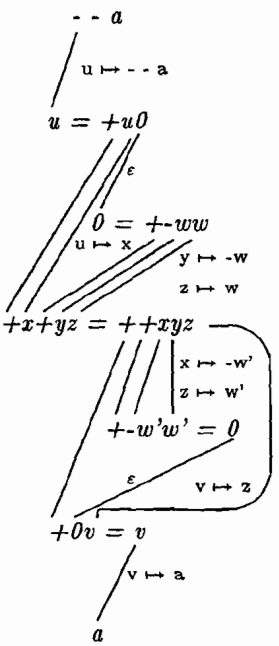
is the graph after the insertion of the equality chain $u = +u0 - +x+yz = ++xyz - +0v = v$. Four subproblems indicated by the dashed lines must be solved and their solutions must be combined to solve the whole problem. The chains to be inserted must have the property that the terms of each arising subproblem have the same top level symbol. In this case: $u - -$, $+ - +$, $+ - +$, and $v - a$



The first and fourth subproblems are solved, the second and third are decomposed again into subproblems, where two of them can be trivially solved. Note that the subproblems $0 = +yz$ and $+xy = 0$ are structurally equal.



The solved subproblems are indicated by lines marked with the corresponding unifier. Two chains can be inserted to solve the nontrivial subproblems of the last graph.



All considered subproblems are solved and the unifiers can be successively combined to derive ε.

In this example only the successful steps of the algorithm are depicted, however as everybody knows who works in the field there is also an enormous amount of useless steps in the search space. The power of an equality prover lies in its facility to avoid such useless steps as much as possible. Even the duplicate steps, like for example the second one with the axiom $+ - ww = 0$ in the above example, should be avoided.

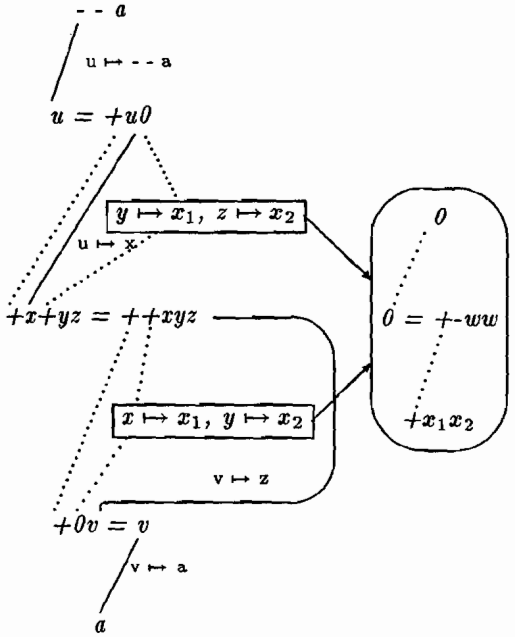
K. Bläsius and V. Lotz [Lo88] used several heuristics in the first implementation of the system and the main power of the program stems from these heuristics. But the results are still unsatisfactory if we consider the standard problems of equality reasoning. Only the first two examples proposed by E. Lusk and R. Overbeek [LO84] could be solved by the program.

In the following we try to illustrate what went wrong using this decomposition technique but first we describe our own extensions of the system.

The global strategy was to switch off all heuristics of K. Bläsius' system to detect and eliminate the weaknesses in the inference mechanism. The first change was the introduction of A. Martelli and U. Montanari's multi equation framework to make the combination with theory unification more feasible, since these algorithms are mostly formulated using such transformation rules. Naturally this change of representation did not give the prover more power. The second and more essential change was to use structure sharing of subproblems to realize the 'subgraph replacement' proposed by K. Bläsius.

We shall now focus on some details of the second change. To use different solutions of structurally identical subproblems at different positions in the graph all subproblems with their graphs are organized in a hashtable. The hashkey is computed from the structure of the two terms of the equality problem. The test for equality of two such pairs of terms (two equality problems) is made efficient by using the same variable, theory-free constant, and theory-free function symbols, that is, $fc_1c_2 = fc_2x$ is structurally the same problem as $fc_2c_1 = fc_1y$ when c_1 and c_2 are Skolem constants (not occurring in a theory) and x and y are variables. This approach is similar to the usual indexing mechanisms in automated theorem proving [OL80,Ohl89]. Everywhere in the graph a "renaming" to the standard representation is stored instead of commonly used subproblems, in the just given example $\{x \mapsto x_1\}$ and $\{c_2 \mapsto c_1, c_1 \mapsto c_2, y \mapsto x_1\}$. Solutions for the subproblems are then simply propagated to all superproblems applying the inverse of the "renaming" to the solutions.

Example 3.3 (Structure Sharing)



This is a variant of the fourth graph of example 3.2. The same (up to renaming) subproblem $0 = +x_1x_2$ occurs at two different positions. The renaming substitutions are depicted in the boxes.

Experiments with the system showed that it often ran into cycles producing arbitrary many unifiers for the same subproblem and propagating them as partial solutions to superproblems. For example it generated the unifiers $\{x \mapsto 0\}, \{x \mapsto -0\}, \{x \mapsto --0\}$ for the problem $x = 0$, that is, it tried to enumerate all unifiers $\{\{x \mapsto 0^n\} \mid n \in N\}$ instead of postponing the computation until nothing better could be done which was normally controlled by the heuristics. Another possibility of avoiding such situations is to use demodulation to reduce all solutions to the simplest ones. The principle of demodulation was introduced and promoted by L. Wos [WRCS67,WOLB84]. Equations are directed and applied to all occurring terms. When using demodulation it is clear that new good demodulators should be computed during the search of a proof and this directly leads to the usage of the Knuth-Bendix completion algorithm.

4 Orientation of Equations in Clause Graphs

Many authors as for example J. Siekmann, [Sie75], A. Bundy [Bun83], and K. Bläsius [Blä86] discussed the complexity of automatically finding a proof for the problem "every group with $x + x = 0$ is commutative" depending on the number of inference steps in the different calculi.

A resolution prover with explicit usage of the equality axioms as stated above based on breadth first search must generate about 10^{21} resolvents to prove this theorem. For a similarly uninformed paramodulation prover the situation is "slightly" better: it "only" has to create approximately 12^{10} ($\approx 6 \cdot 10^{10}$) clauses. This smaller number of steps comes from the fact that paramodulation search trees are not as deep as resolution search trees, but they are much more bushy, and so the amount of reduction is less than hoped for by the inventors of paramodulation.

It is intuitive that an orientation of the equations, that is, their usage in only one direction, leads to another drastic reduction of unnecessary steps. However the reduction is more enormous than someone can imagine, who does not know the Knuth-Bendix completion method. A comparable inference step is to choose a critical pair and incorporate it as a rule. Administrating the critical pairs with a FIFO-strategy which simulates the breadth first search leads to a proof in less than 100 steps. With some simple refinements the number of steps is reduced to 7 (Example 4.2).

This example alone shows that completion is indispensible for equality reasoning and should be placed into the centre of every efficient equality reasoning program. So the question arises why this approach was ignored in almost all theorem proving systems based on the traditional methods of AI.

First of all the application of rewriting seemed to be restricted to some special cases where a canonical rewriting system can be derived. But even if the Knuth-Bendix procedure diverges enough interesting results can be derived as examplified in problems 4.4 and 4.6. With help of the examples we shall demonstrate that most generated clauses are useful lemmas to finally find the proofs for the theorems.

In these cases completion may be superior to the other methods dealing with equality. Many researchers overlooked the capacity for development imbedded in this approach. Meanwhile there are results in handling undirectable equations [BDP87] and using special theory unification and matching algorithms [Sti85,KZ89]. In addition there are attempts to use rewrite systems to construct universal unification algorithms [Kir87] and to integrate this work with conditional equations [Ric83a,Pet83,Kap84,ZR85,JW86,JL87,Rus87,ZK88] but unfortunately all these attempts are not as convincing as the pure method when unit equations are directable.

Another disadvantage of completion not yet mentioned is that completion represents a forward reasoning method without goal and that no possibility exists to distinguish between different abstraction levels. The parameters to be set are only the reduction ordering and the selection strategy to choose critical pairs to be directed.

E. Lusk and R. Overbeek [LO84] published a set of six equality problems without conditions that should be useful to check the power of an equality reasoning procedure. Finishing this section we show the results of the experiments solving the first five examples of E. Lusk and R. Overbeek with a conventional resolution and paramodulation based theorem prover.

Here we have to throw a view on the inards of the Markgraf-Karl system, say the clause graph calculus. More detailed descriptions can be found in [BBB+84,OS89,EOP89,Eis89]. A clause graph consists of a set of clauses, each of them a multi set of literals, and a set of links, which join pairs of literals with unifiable atoms. A link joining a positive and a negative literal is called an R-link (Resolution), while an S-link (Subsumption) joins two literals with the same sign. If the literals incident with a link belong to two different clauses, it is an R2- or S2-link. If both literals belong to the same clause, the link is called R1- or S1-link. In this case it may be that the atoms of the literals are unifiable only after renaming their variables apart, then we speak of a weak link.

The different kinds of links provide immediate access to different kinds of operations involving a given literal occurrence. Most notably, R2-links represent the possible applications of the resolution rule and S1-links indicate factoring. When applying such deduction rules, we have to add to the graph the new clause along with the links connecting the new literals to the just existing graph. If the new literals are instances of ancestor literals already

present in the graph, the new links can be obtained without searching by a simple inheritance process. This inheritance was invented by R. Kowalski [Kow75] and later extended to R1-links by M. Bruynooghe [Bru75]. For a detailed explanation of the mechanism in the MKRP-system see H. J. Ohlbach [Ohl87]. The transfer to S-links is trivial. For new literals that are not obtained by instantiating others, for example the paramodulated literal in a paramodulation step, this form of link inheritance does not work.

In the case of paramodulation there have also been attempts at approaches based on links and inheritance [SW80]. Links to be paramodulated upon do not join literals, they join one side of a positive literal with equality predicate with an arbitrary unifiable term in another literal. They are P2-links if the other literal is in another clause, P1-links if they are in the same. Such a link mechanism was implemented in our system, but unfortunately P-link inheritance can not work as this for R-links because in each resolution or paramodulation step unifiers are applied and therefore completely new terms are generated. Hence our first task was to repair this inheritance mechanism to produce the lacking links. This is simply done by newly generating all P-links.

In addition we made two changes to the theorem prover. The first one concerns the strategy of selecting the links to operate upon. First all demodulating paramodulation links are selected, then possible resolutions are done according to the selected resolution strategy, and the next steps are paramodulation steps corresponding to the generation of critical pairs.

Definition 4.1 (Control Strategy)

```
while empty clause is not derived
  if demodulation P-links exist
     then operate on one of them
     else if R-links exist
             then operate on this link selected by the corresponding selection function
             else if P-links exist
                  then operate on one of them producing the smallest critical pair
                  else error: graph collapsed
```

The second change is for efficiency. When our paramodulation strategy is selected, only these paramodulation links are generated that are applicable in the sense of completion. In this way only P-links are generated, which represent critical pairs. The reduction of generated P-links is illustrated in the following table.

Number of initial links	Wos_1	Wos_2	Wos_3	Wos_4	Wos_5	Wos_6
Using completion	5	1	16	5	12	28
Without completion	38	21	158	68	69	161

In the actual version of the system we lose the main advantage of connection graphs, which is the inheritance mechanism for links, but this idea fails for equality links, because equality operations drastically change the term structure. But with a (not yet implemented) link construction tool based on an indexing mechansim [OL80,Ohl89] we think that the generation of the necessary paramodulation links enhances the advantages of storing this information explicitly. The strategy 4.1 is an obviously incomplete restriction strategy but very useful for many examples in practice where unconditional equations occur.

A first improvement of the method is to handle demodulating P-links separately like all other clause graph reduction rules as for example purity, subsumption, tautolgy, replacement factoring, and replacement resolution. This reduces the number of performed paramodulation steps but not really the time spent for the refutation and in particular it does not really change the strategy.

But of course it can be changed to a complete one such that R- and P-links are considered to have equal rights, that is, the same weight function is applied to both types of links. For example we can choose the link producing the smallest clause and adding its depth in the search space. Such a strategy is selected for the examples 5.7 and 5.8 and given in definition 5.6.

Now we come to the examples. The default reduction ordering for the system is a lexicographic recursive path ordering with the precedence * > - > + > 1 > 0. We changed it for the examples 4.5 and 4.6. The selected

one is explained there. The number of paramodulation steps really performed is not depicted in the protocol and so we give it after the examples in a separate table.

The asterisks in the protocols label the axioms and derived clauses really used in the proof.

The first theorem states that every group with $x + x = 0$ is commutative, and this problem is very trivial using the completion technique.

Example 4.2 (Wos 1)

```
        Set of Axiom Clauses Resulting from Normalization
        =================================================

        A1:   All x:Any + =(x x)
      * A2:   All x,y,z:Any + =(+(+(z y) x) +(z +(y x)))
      * A3:   All x:Any + =(+(0 x) x)
        A4:   All x:Any + =(+(-(x) x) 0)
      * A5:   All x:Any + =(+(x x) 0)


        Set of Theorem Clauses Resulting from Normalization
        ===================================================

      * T6: - =(+(c_2 c_1) +(c_1 c_2))
```

```
Refutation:
===========

A5,1 & A2,1      -->  * P1:    All x,y:Any + =(+(0 y) +(x +(x y)))
P1,1 & A3        -->  * RW2:   All x,y:Any + =(y +(x +(x y)))
A5,1 & RW2,1     -->  * P3:    All x:Any + =(x +(x 0))
A5,1 & A2,1      -->  * P8:    All x,y:Any + =(0 +(y +(x +(y x))))
P8,1 & RW2,1     -->  * P9:    All x,y:Any + =(+(y +(x y)) +(x 0))
P9,1 & P3        -->  * RW10:  All x,y:Any + =(+(y +(x y)) x)
RW10,1 & RW2,1   -->  * P12:   All x,y:Any + =(+(y x) +(x y))
P12,1 & T6,1     -->  * R13:   []
```

```
  q.e.d.
```

The second theorem states that the inverse of a group is an involution, and this problem is trivial too, especially because for a non-commutative group there exists a complete and confluent rewrite system, such that in this theory all purely equational theorems can be solved.

Example 4.3 (Wos 2)

```
        Set of Axiom Clauses Resulting from Normalization
        =================================================

        A1:   All x:Any + =(x x)
      * A2:   All x,y,z:Any + =(+(+(z y) x) +(z +(y x)))
```

```
* A3:   All x:Any + =(+(0 x) x)
* A4:   All x:Any + =(+(-(x) x) 0)


  Set of Theorem Clauses Resulting from Normalization
  ===================================================

* T5: - =(-(-(c_1)) c_1)
```

```
Refutation:
===========

A4,1 & A2,1    -->  * P1:    All x,y:Any + =(+(0 y) +(-(x) +(x y)))
P1,1 & A3      -->  * RW2:   All x,y:Any + =(y +(-(x) +(x y)))
A4,1 & RW2,1   -->  * P4:    All x:Any + =(x +(-(-(x)) 0))
P4,1 & RW2,1   -->  * P8:    All x:Any + =(0 +(-(-(-(x))) x))
P8,1 & RW2,1   -->  * P9:    All x:Any + =(x +(-(-(-(-(x)))) 0))
P9,1 & P4      -->  * RW10:  All x:Any + =(x -(-(x)))
RW10,1 & T5,1  -->  * R11:   □
```

---

q.e.d.

The third example comes from ring theory and is therefore more complicated but not really difficult because the commutativity of addition is not involved and so no undirectable equations must be applied.

Example 4.4 (Wos 3)

```
  Set of Axiom Clauses Resulting from Normalization
  =================================================

  A1:   All x:Any + =(x x)
* A2:   All x,y,z:Any + =(+(+(z y) x) +(z +(y x)))
* A3:   All x:Any + =(+(0 x) x)
* A4:   All x:Any + =(+(-(x) x) 0)
  A5:   All x,y:Any + =(+(y x) +(x y))
  A6:   All x,y,z:Any + =(*(*(z y) x) *(z *(y x)))
* A7:   All x,y,z:Any + =(*(+(z y) x) +(*(z x) *(y x)))
* A8:   All x,y,z:Any + =(*(z +(y x)) +(*(z y) *(z x)))
* A9:   All x:Any + =(*(x x) x)


  Set of Theorem Clauses Resulting from Normalization
  ===================================================

* T10: - =(*(c_1 c_2) *(c_2 c_1))
```

```
Refutation:
==========

A4,1 & A2,1       -->  * P1:    All x,y:Any + =(+(0 y) +(-(x) +(x y)))
P1,1 & A3         -->  * RW2:   All x,y:Any + =(y +(-(x) +(x y)))
A4,1 & RW2,1      -->  * P4:    All x:Any + =(x +(-(-(x)) 0))
P4,1 & RW2,1      -->  * P11:   All x:Any + =(0 +(-(-(-(x))) x))
P11,1 & RW2,1     -->  * P12:   All x:Any + =(x +(-(-(-(-(x)))) 0))
P12,1 & P4        -->  * RW13:  All x:Any + =(x -(-(x)))
P4,1 & RW13       -->  * RW15:  All x:Any + =(x +(x 0))
RW13,1 & A4,1     -->  * P16:   All x:Any + =(+(x -(x)) 0)
P16,1 & A2,1      -->  * P18:   All x,y:Any + =(0 +(y +(x -(+(y x)))))
P18,1 & RW2,1     -->  * P19:   All x,y:Any + =(+(y -(+(x y))) +(-(x) 0))
P19,1 & RW15      -->  * RW20:  All x,y:Any + =(+(y -(+(x y))) -(x))
RW15,1 & A7,1     -->  * P22:   All x,y:Any + =(*(y x) +(*(y x) *(0 x)))
P22,1 & RW2,1     -->  * P23:   All x,y:Any + =(*(0 y) +(-(*(x y)) *(x y)))
P23,1 & A4        -->  * RW24:  All x:Any + =(*(0 x) 0)
A4,1 & A7,1       -->  * P26:   All x,y:Any + =(*(0 y) +(*(-(x) y) *(x y)))
P26,1 & RW24      -->  * RW27:  All x,y:Any + =(0 +(*(-(y) x) *(y x)))
A9,1 & RW27,1     -->  * P28:   All x:Any + =(0 +(*(-(x) x) x))
P28,1 & RW2,1     -->  * P29:   All x:Any + =(x +(-(*(-(x) x)) 0))
P29,1 & RW15      -->  * RW30:  All x:Any + =(x -(*(-(x) x)))
RW30,1 & RW13,1   -->  * P31:   All x:Any + =(*(-(x) x) -(x))
RW13,1 & P31,1    -->  * P34:   All x:Any + =(*(x -(x)) -(-(x)))
P34,1 & RW13      -->  * RW35:  All x:Any + =(*(x -(x)) x)
RW15,1 & A8,1     -->  * P45:   All x,y:Any + =(*(y x) +(*(y x) *(y 0)))
P45,1 & RW2,1     -->  * P46:   All x,y:Any + =(*(y 0) +(-(*(y x)) *(y x)))
P46,1 & A4        -->  * RW47:  All x:Any + =(*(x 0) 0)
A4,1 & A8,1       -->  * P49:   All x,y:Any + =(*(y 0) +(*(y -(x)) *(y x)))
P49,1 & RW47      -->  * RW50:  All x,y:Any + =(0 +(*(y -(x)) *(y x)))
RW35,1 & RW50,1   -->  * P51:   All x:Any + =(0 +(x *(x x)))
P51,1 & A9        -->  * RW52:  All x:Any + =(0 +(x x))
RW52,1 & RW2,1    -->  * P53:   All x:Any + =(x +(-(x) 0))
P53,1 & RW15      -->  * RW54:  All x:Any + =(x -(x))
RW54              -->  * RS55:  All x:Any + =(-(x) x)
RW20,1 & RS55     -->  * RW61:  All x,y:Any + =(+(y +(x y)) x)
A2,1 & RW61,1     -->  * P66:   All x,y,z:Any + =(+(z +(y +(x +(z y)))) x)
RW61,1 & P66,1    -->  * P74:   All x,y,z:Any + =(+(z +(+(y z) +(x y))) x)
P74,1 & A2        -->  * RW75:  All x,y,z:Any + =(+(z +(y +(z +(x y)))) x)
A9,1 & A7,1       -->  * P85:   All x,y:Any + =(+(y x) +(*(y +(y x)) *(x +(y x))))
P85,1 & A8        -->  * RW86:  All x,y:Any + =(+(y x) +(*(y +(y x)) +(*(x y) *(x x))))
RW86,1 & A9       -->  * RW87:  All x,y:Any + =(+(y x) +(*(y +(y x)) +(*(x y) x)))
RW87,1 & A8       -->  * RW88:  All x,y:Any + =(+(y x) +(+(*(y y) *(y x)) +(*(x y) x)))
RW88,1 & A2       -->  * RW89:  All x,y:Any + =(+(y x) +(*(y y) +(*(y x) +(*(x y) x))))
RW89,1 & A9       -->  * RW90:  All x,y:Any + =(+(y x) +(y +(*(y x) +(*(x y) x))))
RW90,1 & RW75,1   -->  * P91:   All x,y:Any + =(+(y +(+(*(x y) x) +(y x))) *(y x))
P91,1 & A2        -->  * RW92:  All x,y:Any + =(+(y +(*(x y) +(x +(y x)))) *(y x))
RW92,1 & RW61     -->  * RW93:  All x,y:Any + =(+(y +(*(x y) y)) *(y x))
RW93,1 & RW61     -->  * RW94:  All x,y:Any + =(*(y x) *(x y))
RW94,1 & T10,1    -->  * R95:   []

------------------------------------------------------------------------------------------------------------

  q.e.d.
```

The fourth example is another special theorem of group theory and in complexity comparable to the third one.

Again no unorientable equation is necessary and therefore no real problem arises. Here an additional feature of the Markgraf-Karl system comes into the game. It detects definitory equations and replaces the definiens at every occurrence by the definiendum. This is sometimes very useful because it reduces the clause set and especially the number of function symbols. Here **A6** is taken as definition for **comm** and so **comm** is completely eliminated by preprocessing operations. For this example we chose the usual Knuth-Bendix ordering for the completion of groups [KB70] such that clause **RW21** is directed from left to right. The precedence is * > - > + > 1 > 0 as above, and the weights are *: 1, +: 1, -: 0, 0: 0, 1: 0. In this way the refutation is found faster, because the search space is smaller.

Example 4.5 (Wos 4)

```
          Set of Axiom Clauses Resulting from Normalization
          =================================================

        * A1:   All x:Any + =(x x)
        * A2:   All x,y,z:Any + =(+(+(z y) x) +(z +(y x)))
        * A3:   All x:Any + =(+(0 x) x)
        * A4:   All x:Any + =(+(-(x) x) 0)
        * A5:   All x:Any + =(+(+(x x) x) 0)
        * A6:   All x,y:Any + =(comm(y x) +(y +(x +(-(y) -(x)))))


          Initial Operations on Axioms
          ============================

A5,1 & A2 --> * RW1: All x:Any + =(+(x +(x x)) 0)


          Set of Theorem Clauses Resulting from Normalization
          ===================================================

        * T7: - =(comm(comm(c_1 c_2) c_2) 0)


          Initial Operations on Theorems
          ==============================

T7,1 & A6  --> * RW2: - =(+(comm(c_1 c_2) +(c_2 +(-(comm(c_1 c_2)) -(c_2)))) 0)
RW2,1 & A6 --> * RW3: - =(+(+(c_1 +(c_2 +(-(c_1) -(c_2))))
                            +(c_2 +(-(+(c_1 +(c_2 +(-(c_1) -(c_2))))) -(c_2))))
                          0)
RW3,1 & A2 --> * RW4: - =(+(c_1 +(+(c_2 +(-(c_1) -(c_2)))
                                   +(c_2 +(-(+(c_1 +(c_2 +(-(c_1) -(c_2))))) -(c_2)))))
                          0)

RW4,1 & A2 --> * RW5: - =(+(c_1 +(c_2 +(+(-(c_1) -(c_2))
                                        +(c_2 +(-(+(c_1 +(c_2 +(-(c_1) -(c_2))))) -(c_2))))))
                          0)
RW5,1 & A2 --> * RW6: - =(+(c_1 +(c_2 +(-(c_1) +(-(c_2) +(c_2 +(-(+(c_1 +(c_2 +(-(c_1) -(c_2)))))
                                                                  -(c_2)))))))
                          0)

-------------------------------------------------------------------------------------------------
```

```
Refutation:
===========

A4,1 & A2,1      -->  * P7:    All x,y:Any + =(+(0 y) +(-(x) +(x y)))
P7,1 & A3        -->  * RW8:   All x,y:Any + =(y +(-(x) +(x y)))
RW6,1 & RW8      -->  * RW9:   - =(+(c_1 +(c_2 +(-(c_1) +(-(+(c_1 +(c_2 +(-(c_1) -(c_2))))) -(c_2)))))
                                 0)
A4,1 & RW8,1     -->  * P14:   All x:Any + =(x +(-(-(x)) 0))
RW1,1 & RW8,1    -->  * P15:   All x:Any + =(+(x x) +(-(x) 0))
RW1,1 & A2,1     -->  * P16:   All x,y:Any + =(+(0 y) +(x +(+(x x) y)))
P16,1 & A2       -->  * RW17:  All x,y:Any + =(+(0 y) +(x +(x +(x y))))
RW17,1 & A3      -->  * RW18:  All x,y:Any + =(y +(x +(x +(x y))))
RW1,1 & RW18,1   -->  * P19:   All x:Any + =(x +(x 0))
P14,1 & P19      -->  * RW20:  All x:Any + =(x -(-(x)))
P15,1 & P19      -->  * RW21:  All x:Any + =(+(x x) -(x))
RW1,1 & RW21     -->  * RW22:  All x:Any + =(+(x -(x)) 0)
RW22,1 & A2,1    -->  * P27:   All x,y:Any + =(+(0 y) +(x +(-(x) y)))
P27,1 & A3       -->  * RW28:  All x,y:Any + =(y +(x +(-(x) y)))
A2,1 & RW22,1    -->  * P32:   All x,y:Any + =(+(y +(x -(+(y x)))) 0)
P32,1 & RW8,1    -->  * P33:   All x,y:Any + =(+(y -(+(x y))) +(-(x) 0))
P33,1 & P19      -->  * RW34:  All x,y:Any + =(+(y -(+(x y))) -(x))
RW21,1 & A2,1    -->  * P36:   All x,y:Any + =(+(-(y) x) +(y +(y x)))
RW34,1 & RW8,1   -->  * P38:   All x,y:Any + =(-(+(y x)) +(-(x) -(y)))
RW9,1 & P38      -->  * RW40:  - =(+(c_1 +(c_2 +(-(c_1) +(+(-(+(c_2 +(-(c_1) -(c_2)))) -(c_1))
                                                          -(c_2)))))
                                 0)
RW40,1 & P38     -->  * RW41:  - =(+(c_1 +(c_2 +(-(c_1) +(+(+(-(+(-(c_1) -(c_2))) -(c_2)) -(c_1))
                                                          -(c_2)))))
                                 0)
RW41,1 & P38     -->  * RW42:  - =(+(c_1 +(c_2 +(-(c_1) +(+(+(+(-(-(c_2)) -(-(c_1))) -(c_2)) -(c_1))
                                                          -(c_2)))))
                                 0)
RW42,1 & RW20    -->  * RW43:  - =(+(c_1 +(c_2 +(-(c_1) +(+(+(+(-(-(c_2)) c_1) -(c_2)) -(c_1))
                                                          -(c_2)))))
                                 0)
RW43,1 & RW20    -->  * RW44:  - =(+(c_1 +(c_2 +(-(c_1) +(+(+(+(c_2 c_1) -(c_2)) -(c_1)) -(c_2))))) 0)
RW44,1 & A2      -->  * RW45:  - =(+(c_1 +(c_2 +(-(c_1) +(+(+(c_2 +(c_1 -(c_2))) -(c_1)) -(c_2))))) 0)
RW45,1 & A2      -->  * RW46:  - =(+(c_1 +(c_2 +(-(c_1) +(+(c_2 +(+(c_1 -(c_2)) -(c_1))) -(c_2))))) 0)
RW46,1 & A2      -->  * RW47:  - =(+(c_1 +(c_2 +(-(c_1) +(+(c_2 +(c_1 +(-(c_2) -(c_1)))) -(c_2))))) 0)
RW47,1 & A2      -->  * RW48:  - =(+(c_1 +(c_2 +(-(c_1) +(c_2 +(+(c_1 +(-(c_2) -(c_1))) -(c_2)))))) 0)
RW48,1 & A2      -->  * RW49:  - =(+(c_1 +(c_2 +(-(c_1) +(c_2 +(c_1 +(+(-(c_2) -(c_1)) -(c_2))))))) 0)
RW49,1 & A2      -->  * RW50:  - =(+(c_1 +(c_2 +(-(c_1) +(c_2 +(c_1 +(-(c_2) +(-(c_1) -(c_2)))))))) 0)
A2,1 & RW21,1    -->  * P51:   All x,y:Any + =(+(y +(x +(y x))) -(+(y x)))
P51,1 & P38      -->  * RW52:  All x,y:Any + =(+(y +(x +(y x))) +(-(x) -(y)))
RW52,1 & P36,1   -->  * P56:   All x,y:Any + =(+(-(y) +(x +(y x))) +(y +(-(x) -(y))))
RW20,1 & P56,1   -->  * P57:   All x,y:Any + =(+(y +(x +(-(y) x))) +(-(y) +(-(x) -(-(y)))))
P57,1 & RW20     -->  * RW58:  All x,y:Any + =(+(y +(x +(-(y) x))) +(-(y) +(-(x) y)))
RW50,1 & RW58    -->  * RW59:  - =(+(c_1 +(c_2 +(-(c_1) +(c_2 +(-(c_1) +(-(-(c_2)) c_1))))))) 0)
RW59,1 & RW20    -->  * RW60:  - =(+(c_1 +(c_2 +(-(c_1) +(c_2 +(-(c_1) +(c_2 c_1))))))) 0)
P36,1 & A2,1     -->  * P61:   All x,y,z:Any + =(+(-(+(z y)) x) +(z +(y +(+(z y) x))))
P61,1 & A2       -->  * RW62:  All x,y,z:Any + =(+(-(+(z y)) x) +(z +(y +(z +(y x)))))
RW62,1 & P38     -->  * RW63:  All x,y,z:Any + =(+(+(-(z) -(y)) x) +(y +(z +(y +(z x)))))
RW63,1 & A2      -->  * RW64:  All x,y,z:Any + =(+(-(z) +(-(y) x)) +(y +(z +(y +(z x)))))
RW60,1 & RW64    -->  * RW65:  - =(+(c_1 +(c_2 +(-(c_2) +(-(-(c_1)) c_1)))) 0)
RW65,1 & RW20    -->  * RW66:  - =(+(c_1 +(c_2 +(-(c_2) +(c_1 c_1)))) 0)
RW66,1 & RW21    -->  * RW67:  - =(+(c_1 +(c_2 +(-(c_2) -(c_1)))) 0)
```

```
RW67,1 & RW28   -->  * RW68:  - =(+(c_1 -(c_1)) 0)
RW68,1 & RW22   -->  * RW69:  - =(0 0)
RW69,1 & A1,1   -->  * R70:   []
```

---

q.e.d.

The fifth example is one in the not so widely known theory of ternary algebra and we have to say a few words about the problem and our proof. In most cases the theory is given with an additional axiom: a right inverse `*(x y -(y)) = x`. But only one of the inverses is necessary. Normally a Knuth-Bendix reduction ordering, which we used for this example, sets the parentheses rightassociative so that for example a(b(c(d e))) is the normal form and not (((a b)c)d)e. In this case omitting the left inverse causes no difficulties and the theorem can be proved despite of the divergence of the completion algorithm. But omitting the right inverse induces trouble, the left side of the inverse rule cannot be unified with the left side of the equation `P20`. So the other way of setting the parentheses must be chosen to find a proof with this method. It is a commonly used technique to define orderings from left to right or right to left for different operators in term rewriting. Using the left to right ordering and unfailing completion such that the necessary derived unorientable equations can be applied in both directions, increases the search space enormously such that MKRP does not find the solution.

Example 4.6 (Wos 5)

```
Set of Axiom Clauses Resulting from Normalization
=================================================

  A1:   All x:Any + =(x x)
* A2:   All x,y,z,u,v:Any + =(*(*(v u z) y *(v u x)) *(v u *(z y x)))
* A3:   All x,y:Any + =(*(y x x) x)
* A4:   All x,y:Any + =(*(y y x) y)
* A5:   All x,y:Any + =(*(-(y) y x) x)


Set of Theorem Clauses Resulting from Normalization
===================================================

* T6: - =(*(c_1 -(c_1) c_2) c_2)
```

```
Refutation:
===========

A4,1 & A2,1     -->  * P1:    All x,y,z,u:Any + =(*(u z *(u u y)) *(u u *(x z y)))
P1,1 & A4       -->  * RW2:   All x,y,z:Any + =(*(z y *(z z x)) z)
RW2,1 & A4      -->  * RW3:   All x,y:Any + =(*(y x y) y)
A3,1 & A2,1     -->  * P7:    All x,y,z,u:Any + =(*(*(u z y) x z) *(u z *(y x z)))
A5,1 & P7,1     -->  * P8:    All x,y,z:Any + =(*(*(z y -(x)) x y) *(z y y))
P8,1 & A3       -->  * RW9:   All x,y,z:Any + =(*(*(z y -(x)) x y) y)
A4,1 & P7,1     -->  * P10:   All x,y,z:Any + =(*(*(z y x) x y) *(z y x))
P10,1 & RW9,1   -->  * P11:   All x,y,z:Any + =(*(*(z -(y) x) y x) x)
RW3,1 & A2,1    -->  * P20:   All x,y,z,u:Any + =(*(*(u z y) x u) *(u z *(y x u)))
A4,1 & P20,1    -->  * P23:   All x,y,z:Any + =(*(*(z y x) x z) *(z y x))
P23,1 & P10,1   -->  * P25:   All x,y,z:Any + =(*(*(z y x) z x) *(*(z y x) x z))
P25,1 & P23     -->  * RW26:  All x,y,z:Any + =(*(*(z y x) z x) *(z y x))
```

```
P11,1 & RW26,1  -->  * P27:   All x,y:Any + =(y *(x -(x) y))
P27,1 & T6,1    -->  * R28:   []
```

```
q.e.d.
```

E. Lusk and R. Overbeek mentioned a sixth problem, which is probably the most famous one in equality reasoning: "every ring with $x^3 = x$ is commutative." Many authors as for example M. Stickel [Sti84] and D. Kapur [KZ89] focused on it. They used special techniques to solve it and other related problems in the family $x^n = x$, especially they used completion modulo AC-unification. In particular D. Kapur developed a special algorithm to handle these problems very efficiently. Without AC-unification it is not feasible to solve this example. We shall extend our theorem prover to use AC1-unification with constraints [KK89].

The combination of theory unification algorithms and the Knuth-Bendix procedure seems the most promising way to handle unit (unconditional) equations.

Now we come to the premised table of steps and we give the ratio **number of performed steps / number of proof steps** in a second column. Numbers smaller than one stem from the reduction steps which are not counted as performed paramodulations, because they are in some sense "deterministic". The third line gives the ratio when only completion steps in the proof are counted. The table is another hint that the completion process can be seen as a very straightforward lemma generation. With a corresponding selection function and a lookahead that should be more efficient than ours, there are almost all produced clauses useful for the proof.

	Wos_1	Wos_2	Wos_3	Wos_4	Wos_5	Wos_6
Steps	6	7	41	18	20	∞
Ratio with rewrites	0.75	1	0.89	0.34	1.43	∞
Ratio without rewrites	1.2	1.75	1.78	1.29	2.22	∞

The numbers of the clauses in the protocol are not related to the numbers of clauses generated during the proof, they get there numbers in the protocol module due to some mystery.

As the last point in this section we give a table to compare the runtimes for the given examples in K. Bläsius' system and in the Markgraf-Karl system. The time is measured in seconds and the computations were done on a Symbolics 36xx. ∞ means that the program cannot solve the problem.

	Wos_1	Wos_2	Wos_3	Wos_4	Wos_5	Wos_6
Bläsius	89	40	∞	∞	∞	∞
MKRP	25	18	546	245	312	∞

5 Additional Mechanisms, Orientation of Clauses

Now the equality reasoning mechanism is combined with some other facilities of the MKRP-system as for example splitting, which makes it possible to formulate some problems in a more natural way, because the parts of the proof are proved separately. One example is 5.1.

Example 5.1 (Splitting)
In a group are equivalent:

1. $x + y = y + x$
2. $(x + y) + (y + x) = (x + x) + (y + y)$
3. $-(x + y) = -(x) + -(y)$

The three parts of a circular implication can be given as one formula and are then proved independently.

Another feature of our theorem prover are sorts and they cause no practical trouble as long as we take care that the ordering is compatible with the sort structure. Of course there is an abundance of literature describing the combination of sorts and equality reasoning methods. J. Goguen, J.-P. Jouannaud, and J. Meseguer [GJM85], A. Dick [Dic85], K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer [FGJM85], M. Bidoit, and M. Glaudel [BG85], and J.-P. Jouannaud, and P. Lescanne [JL87] have a view from program specification on rewriting and sorts, the viewpoint of R. Cunningham, and A. Dick [CD85], J. Goguen, and J. Meseguer [GM85], G. Smolka, and collegues [SNMG87], as well as the works of J. Gallier, and T. Isakowitz [GI88], and M. Schmidt-Schauß[SS88] are more theoretical. Especially [SS88] must be considered when fully integrating sorts and rewriting. But there is also a lot of open questions in this area, in particular it is not clear how strong the restriction on the correlation of the reduction ordering and the sort hierarchy must be.

One of the most powerful tools in MKRP is its dedicated clause graph reduction facility [Prä85,EOP89] and especially the subsumption rule is very useful in the equality reasoning context. We implemented an extended form, which allows a lookahead of demodulation steps such that a lot of unnecessary link generations could be suppressed. But there remains an enormous overhead caused by the generation of these links.

As mentioned above simple completion and demodulation cannot be the unique mechanism to grab the problem of handling the equality predicate automatically. The first implication ($1 \Rightarrow 2$) of example 5.1 is a suitable instance of such a problem. To prove 2 nothing must be done but first switching the middle pair of identifiers and then the right pair. But associativity can only be applied left to right and so we have an equation $x + (y + (y + x)) = x + (x + (y + y))$ and a relatively complicated proof using unfailing completion with 26 completion and demodulation steps is generated by our theorem prover. Using AC-unification the proof consists of a trivial unification step, which only has to state that the two terms are equal. This shows again that theory unification is a powerful tool in combination with rewriting.

Now we turn to the problem of conditional equations. In most cases they occur together with unit equations that can be handled by completion and rewriting. So one strategy would be to generate all "good and necessary" rewrite rules and then to use heuristics when applying the conditional equations.

We shall demonstrate this principle with the following two examples taken from the theory of commutative, zero divisor free rings. The ring axioms are given as usual and the property that the ring has no zero divisors is expressed by a conditional equation $\forall x, y : x \cdot y = 0 \Rightarrow x = 0 \vee y = 0$, which can be transformed into a clause $x \cdot y \neq 0 \vee x = 0 \vee y = 0$. This is the clause in the upper right corner of the example 5.3.

First of all we give a proof of a human mathematician and then we illustrate in example 5.3 the infeasibility of paramodulation and rewriting alone for such really simple problems.

Example 5.2 (Zero Divisor Free Ring, Cancellation, Human Proof)

Let *a)* $(R, +, \cdot, 0, 1)$ *be a commutative ring with 1 and*
b) $\forall x, y : x \cdot y = 0 \Rightarrow x = 0 \vee y = 0$ *(zero divisor free)*

then $\forall x, y, z : x \cdot z = y \cdot z \wedge z \neq 0 \Rightarrow x = y$ *(cancellation)*

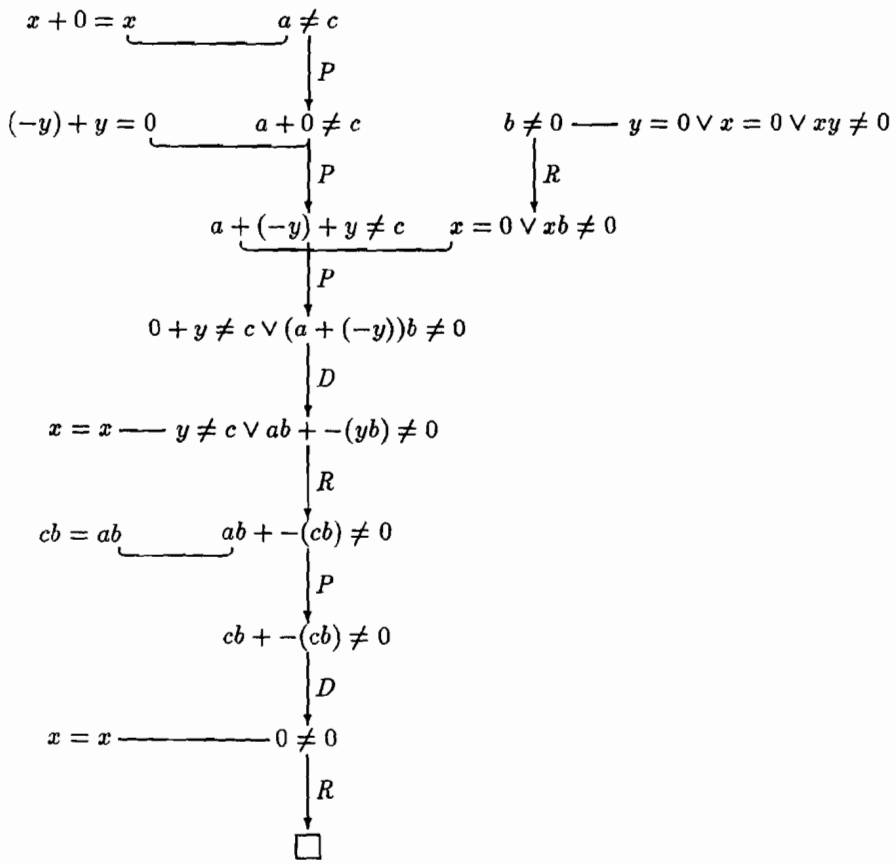
Proof: *Let* $x, y, z \in R$ *and* $x \cdot z = y \cdot z \wedge z \neq 0$
$\Rightarrow x \cdot z - y \cdot z = (x - y) \cdot z = 0$
$\Rightarrow x - y = 0$ *because* $z \neq 0$ *and b)*
$\Rightarrow x = y$
q.e.d.

Next we try to depict a paramodulation proof for this example in form of a graph where the proof steps are labeled with D for demodulation, P for paramodulation, and R for resolution. Sometimes a D-arrow stands for multiple applications of demodulation rules. The proof is hand-made with the following heuristics in mind:

Use the human proof as orientation and make the proof as linear as possible and begin the linear chain with a clause in a rather small set of support, that is, not the whole theorem but just an essential part of it. The negated and Skolemized theorem consists of three clauses $cb = ab$, $b \neq 0$, and $a \neq c$ with the Skolem constants a, b, and c. The most restricted set of support consists of just the conclusion of the theorem $a \neq c$ and this should be the nucleus of our linear proof. The first action consisting of two paramodulation steps is to expand one side of the inequality by subtracting and adding the same thing, this "something" is intended to become a "c" so that the literal is $c \neq c$ and can be resolved away. This goal can almost be achieved by applying a rewrite rule (derived via a resolution step at the right hand side of the picture) to make $a + (-y)$ to 0 but we just have a conditional rewrite rule and so we introduce a new literal. The next two steps (D and R) are done to activate our intention and to remove the $c \neq c$ literal. What remains to be done is to eliminate the newly introduced literal, which is done straightforwardly by applying a structurally very simple equation (due to its lack of variables) and demodulating until resolution to the empty clause is possible.

This proof seems to be simple but there are essential disadvantages: Rewrite rules are used in the reverse direction ($x + 0 = x$, $(-y) + y = 0$), paramodulation with variables is used ($x + 0 = x$), associativity is used implicitly in both directions ($a + ((-y) + y) = (a + (-y)) + y$), and a partially completed set of axioms is used (right identity when it is not given).

Example 5.3 (Zero Divisor Free Ring, Cancellation, Paramodulation Proof)



Secondly we illustrate similar problems for a second example in the same theory. Again we begin with the human proof.

Example 5.4 (Zero Divisor Free Ring, Square, Human Proof)

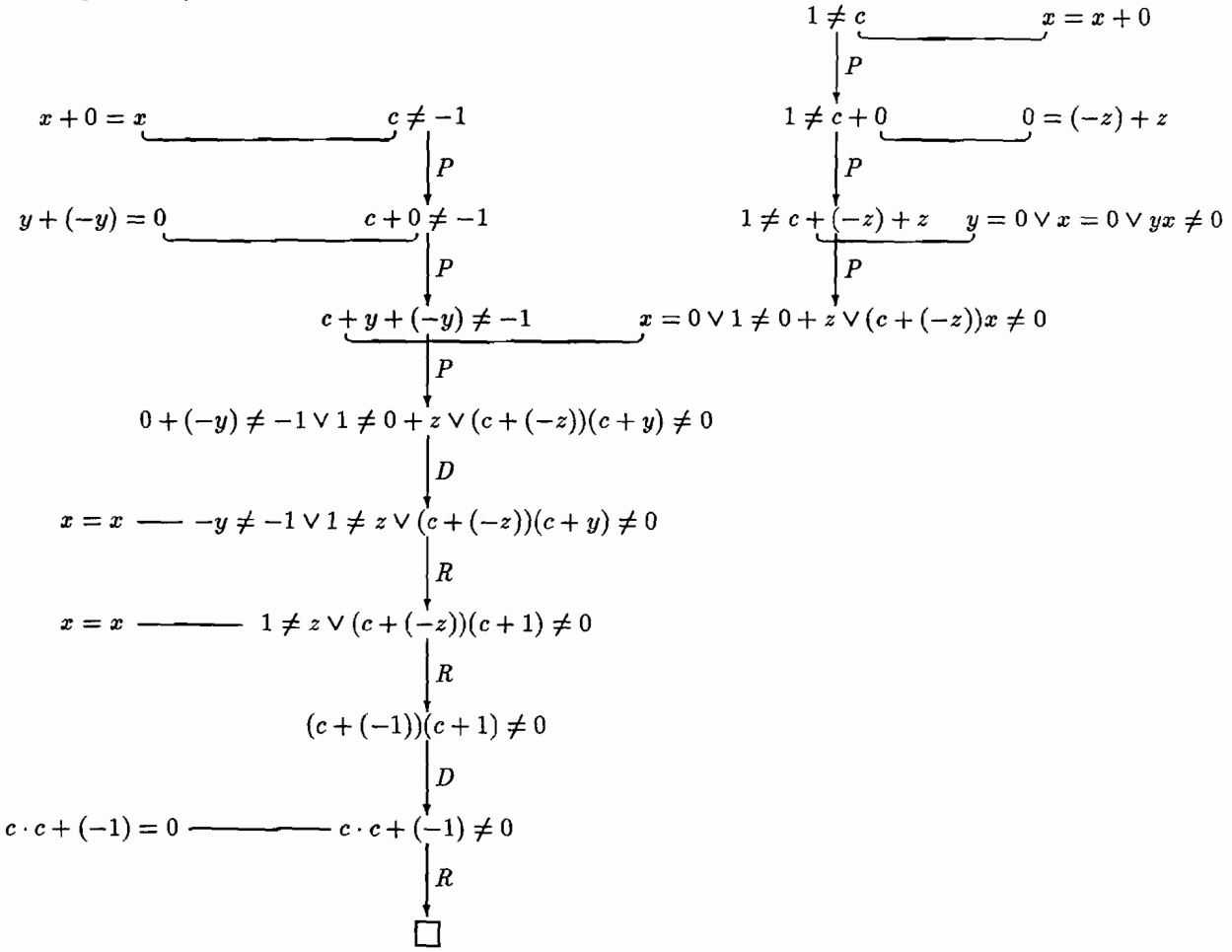
Let *a)* $(R,+,\cdot,0,1)$ *be a commutative ring with 1 and*
b) $\forall x,y : x \cdot y = 0 \Rightarrow x = 0 \vee y = 0$ *(zero divisor free)*

then $\forall x : x^2 - 1 = 0 \Rightarrow x = 1 \vee x = -1$

Proof: *Let* $x \in R$ *and* $x^2 - 1 = 0$
$\Rightarrow 0 = x^2 - 1 = (x-1) \cdot (x+1)$
$\Rightarrow x - 1 = 0 \vee x + 1 = 0$ *because b)*
$\Rightarrow x = 1 \vee x = -1$

The paramodulation proof is similar to the first one. Let us restrict our attention to the main differences: The trick with the expansion and application of conditional $x \rightarrow 0$ is applied twice. It is used in the last demodulation sequence, which additionally is very complicated relative to the one in the other proof. Another difference is that a further completed system of rewrite rules is needed, that is, more completion steps must be performed, and it cannot be decided in advance how many completion steps must be done really.

Example 5.5 (Zero Divisor Free Ring, Square, Paramodulation Proof)



Now, what can be concluded from these examples? The intuitive way (linear paramodulation with rewriting) seems to lead astray, it perverts the human proof to one as complicated as possible and not as machine oriented as necessary. This naive approach to combine heuristics and rewriting is not really convincing. An approach using conditional rewriting and completion is better. G. Peterson [Pet83] was the first who developped a resolution and paramodulation calculus which reduces to the Knuth-Bendix algorithm when only given unit equality axioms and theorems. M. Rusinowitch [HR86,Rus87] extended this work such that only maximal literals of the clauses must be considered for paramodulation. G. Peterson's as well as M. Rusinowitch's approaches only allow very restricted reduction orderings and only demodulations by unit reduction rules. H. Zhang and D. Kapur [ZK88] extended it to more orderings and contextual rewriting. Next we present the proofs found by Markgraf-Karl for our two examples using their method and strategy 5.6.

Definition 5.6 (Advanced Control Strategy)

```
while empty clause is not derived
  if R- or P-links exist
     then select the minimal link according to
               ((if focus on unit clauses and not both clauses are units
                    then punishment factor
                    else 0
                + (link_depth_weight * depth_of_link)
                + (lookahead of size of clause))
               * if both parents are units
                    then reward factor
                    else 1)
          operate on it
     else error: graph collapsed
```

Using the Zhang-Kapur method we drastically change the resolution strategy because only on links joining maximal literals can be resolved or paramodulated. In this way no set-of-support or linear strategy is complete and in fact Markgraf-Karl now has difficulties to solve problems which were solved before. One main task in the future is to avoid this disadvantage.

Example 5.7 (Zero Divisor Free Ring, Cancellation, MKRP Proof)

```
          Formulae given to the editor
          ============================

Axioms:   * RING *
          ALL X,Y,Z +(+(X Y) Z) = +(X +(Y Z))
          ALL X +(0 X) = X
          ALL X +(-(X) X) = 0
          ALL X,Y,Z *(*(X Y) Z) = *(X *(Y Z))
          ALL X,Y,Z *(X +(Y Z)) = +(*(X Y) *(X Z))
          ALL X,Y,Z *(+(Y Z) X) = +(*(Y X) *(Z X))
          * WITH ONE *
          ALL X *(1 X) = X
          ALL X *(X 1) = X
          * ZERO DIVISOR FREE *
          ALL X,Y *(X Y) = 0 IMPL X = 0 OR Y = 0
```

```
Theorems: * CANCELLATION *
          ALL X,Y,Z *(X Y) = *(Z Y) AND NOT (Y = 0) IMPL X = Z


          Set of Axiom Clauses Resulting from Normalization
          =================================================

        * A1:   All x:Any + =(x x)
        * A2:   All x,y,z:Any + =(+(+(z y) x) +(z +(y x)))
        * A3:   All x:Any + =(+(0 x) x)
        * A4:   All x:Any + =(+(-(x) x) 0)
        * A5:   All x,y,z:Any + =(*(*(z y) x) *(z *(y x)))
        * A6:   All x,y,z:Any + =(*(z +(y x)) +(*(z y) *(z x)))
        * A7:   All x,y,z:Any + =(*(+(z y) x) +(*(z x) *(y x)))
        * A8:   All x:Any + =(*(1 x) x)
        * A9:   All x:Any + =(*(x 1) x)
        * A10:  All x,y:Any - =(*(y x) 0)  + =(y 0)  + =(x 0)


          Set of Theorem Clauses Resulting from Normalization
          ===================================================

        * T11: + =(*(c_2 c_1) *(c_3 c_1))
        * T12: - =(c_1 0)
        * T13: - =(c_2 c_3)
```

```
Refutation:
===========

A4,1 & A2,1        -->  * P1:     All x,y:Any + =(+(0 y) +(-(x) +(x y)))
P1,1 & A3          -->  * RW2:    All x,y:Any + =(y +(-(x) +(x y)))
A4,1 & RW2,1       -->  * P4:     All x:Any + =(x +(-(-(x)) 0))
P4,1 & RW2,1       -->  * P10:    All x:Any + =(0 +(-(-(-(x))) x))
P10,1 & RW2,1      -->  * P11:    All x:Any + =(x +(-(-(-(-(x)))) 0))
P11,1 & P4         -->  * RW12:   All x:Any + =(x -(-(x)))
P4,1 & RW12        -->  * RW14:   All x:Any + =(x +(x 0))
RW12,1 & A4,1      -->  * P15:    All x:Any + =(+(x -(x)) 0)
A3,1 & A6,1        -->  * P16:    All x,y:Any + =(*(y x) +(*(y 0) *(y x)))
A9,1 & P16,1       -->  * P17:    All x:Any + =(*(x 1) +(*(x 0) x))
P17,1 & A9         -->  * RW18:   All x:Any + =(x +(*(x 0) x))
RW14,1 & RW18,1    -->  * P19:    + =(0 *(0 0))
A3,1 & A7,1        -->  * P21:    All x,y:Any + =(*(y x) +(*(0 x) *(y x)))
A8,1 & P21,1       -->  * P22:    All x:Any + =(*(1 x) +(*(0 x) x))
P22,1 & A8         -->  * RW23:   All x:Any + =(x +(*(0 x) x))
RW23,1 & RW2,1     -->  * P24:    All x:Any + =(x +(-(*(0 x)) x))
P19,1 & A5,1       -->  * P28:    All x:Any + =(*(0 x) *(0 *(0 x)))
P28,1 & P24,1      -->  * P29:    All x:Any + =(*(0 x) +(-(*(0 x)) *(0 x)))
P29,1 & A4         -->  * RW30:   All x:Any + =(*(0 x) 0)
P28,1 & RW30       -->  * RW31:   All x:Any + =(*(0 x) *(0 0))
RW31,1 & RW30      -->  * RW32:   All x:Any + =(*(0 x) 0)
A4,1 & A7,1        -->  * P82:    All x,y:Any + =(*(0 y) +(*(-(x) y) *(x y)))
P82,1 & RW32       -->  * RW83:   All x,y:Any + =(0 +(*(-(y) x) *(y x)))
RW83,1 & RW2,1     -->  * P87:    All x,y:Any + =(*(y x) +(-(*(-(y) x)) 0))
P87,1 & RW14       -->  * RW88:   All x,y:Any + =(*(y x) -(*(-(y) x)))
```

```
RW88,1 & A4,1      -->  * P97:    All x,y:Any + =(+(*(y x) *(-(y) x)) 0)
P97,1 & RW2,1      -->  * P98:    All x,y:Any + =(*(-(y) x) +(-(*(y x)) 0))
P98,1 & RW14       -->  * RW99:   All x,y:Any + =(*(-(y) x) -(*(y x)))
A7,1 & A10,1       -->  * P154:   All x,y,z:Any - =(+(*(z y) *(x y)) 0)  + =(+(z x) 0)  + =(y 0)
T11,1 & P154,1     -->  * P155:   All x:Any - =(+(*(c_2 c_1) *(x c_1)) 0)  + =(+(c_3 x) 0)  + =(c_1 0)
P155,3 & T12,1     -->  * R156:   All x:Any - =(+(*(c_2 c_1) *(x c_1)) 0)  + =(+(c_3 x) 0)
RW99,1 & R156,1    -->  * P167:   All x:Any - =(+(*(c_2 c_1) -(*(x c_1))) 0) + =(+(c_3 -(x)) 0)
P15,1 & P167,1     -->  * P168:   - =(0 0)  + =(+(c_3 -(c_2)) 0)
P168,1 & A1,1      -->  * R169:   + =(+(c_3 -(c_2)) 0)
R169,1 & RW2,1     -->  * P170:   + =(-(c_2) +(-(c_3) 0))
P170,1 & RW14      -->  * RW171:  + =(-(c_2) -(c_3))
RW171,1 & RW12,1   -->  * P172:   + =(c_3 -(-(c_2)))
P172,1 & RW12      -->  * RW173:  + =(c_3 c_2)
RW173,1 & T13,1    -->  * R174:   []
```

---

q.e.d.

The second example is more difficult and needs a very long run time, a further detailed selection function, or human support. The difficulty stems from the fact that for all three literals in the formula specifying "zero divisor free" equality reasoning steps are necessary to resolve the literals away, whereas in example 5.7 one literal can be resolved without preliminary paramodulation steps (R156). The problem occurs when deriving P237, which is very large compared to other clauses derivable in this state of the refutation process and so the program first selects all smaller and useless clauses before operating on the decisive link.

Example 5.8 (Zero Divisor Free Ring, Square, MKRP Proof)

```
          Formulae given to the editor
          ============================

Axioms:   * RING *
          ALL X,Y,Z +(+(X Y) Z) = +(X +(Y Z))
          ALL X +(0 X) = X
          ALL X +(-(X) X) = 0
          ALL X,Y,Z *(*(X Y) Z) = *(X *(Y Z))
          ALL X,Y,Z *(X +(Y Z)) = +(*(X Y) *(X Z))
          ALL X,Y,Z *(+(Y Z) X) = +(*(Y X) *(Z X))
          * WITH ONE *
          ALL X *(1 X) = X
          ALL X *(X 1) = X
          * ZERO DIVISOR FREE *
          ALL X,Y *(X Y) = 0 IMPL X = 0 OR Y = 0

Theorems: * ZEROS *
          ALL X +(*(X X) -(1)) = 0 IMPL X = 1 OR X = -(1)


          Set of Axiom Clauses Resulting from Normalization
          =================================================

          A1:    All x:Any + =(x x)
        * A2:    All x,y,z:Any + =(+(+(z y) x) +(z +(y x)))
        * A3:    All x:Any + =(+(0 x) x)
        * A4:    All x:Any + =(+(-(x) x) 0)
```

```
  A5:   All x,y,z:Any + =(*(*(z y) x) *(z *(y x)))
* A6:   All x,y,z:Any + =(*(z +(y x)) +(*(z y) *(z x)))
* A7:   All x,y,z:Any + =(*(+(z y) x) +(*(z x) *(y x)))
* A8:   All x:Any + =(*(1 x) x)
* A9:   All x:Any + =(*(x 1) x)
* A10:  All x,y:Any - =(*(y x) 0)  + =(y 0)  + =(x 0)


  Set of Theorem Clauses Resulting from Normalization
  ===================================================

* T11: + =(+(*(c_1 c_1) -(1)) 0)
* T12: - =(c_1 1)
* T13: - =(c_1 -(1))
```

Refutation:
==========

```
A4,1 & A2,1       -->  * P1:     All x,y:Any + =(+(0 y) +(-(x) +(x y)))
P1,1 & A3         -->  * RW2:    All x,y:Any + =(y +(-(x) +(x y)))
A4,1 & RW2,1      -->  * P4:     All x:Any + =(x +(-(-(x)) 0))
T11,1 & A2,1      -->  * P10:    All x:Any + =(+(0 x) +(*(c_1 c_1) +(-(1) x)))
P10,1 & A3        -->  * RW11:   All x:Any + =(x +(*(c_1 c_1) +(-(1) x)))
A4,1 & RW11,1     -->  * P12:    + =(1 +(*(c_1 c_1) 0))
RW2,1 & RW11,1    -->  * P13:    All x:Any + =(+(1 x) +(*(c_1 c_1) x))
P12,1 & P13       -->  * RW16:   + =(1 +(1 0))
RW16,1 & A6,1     -->  * P17:    All x:Any + =(*(x 1) +(*(x 1) *(x 0)))
P17,1 & A9        -->  * RW18:   All x:Any + =(*(x 1) +(x *(x 0)))
RW18,1 & A9       -->  * RW19:   All x:Any + =(x +(x *(x 0)))
RW19,1 & RW2,1    -->  * P21:    All x:Any + =(*(x 0) +(-(x) x))
P21,1 & A4        -->  * RW22:   All x:Any + =(*(x 0) 0)
RW19,1 & RW22     -->  * RW23:   All x:Any + =(x +(x 0))
P4,1 & RW23       -->  * RW24:   All x:Any + =(x -(-(x)))
P13,1 & RW23,1    -->  * P25:    + =(*(c_1 c_1) +(1 0))
P25,1 & RW23      -->  * RW26:   + =(*(c_1 c_1) 1)
RW24,1 & A4,1     -->  * P28:    All x:Any + =(+(x -(x)) 0)
A4,1 & A6,1       -->  * P32:    All x,y:Any + =(*(y 0) +(*(y -(x)) *(y x)))
P32,1 & RW22      -->  * RW33:   All x,y:Any + =(0 +(*(y -(x)) *(y x)))
RW33,1 & RW2,1    -->  * P42:    All x,y:Any + =(*(y x) +(-(*(y -(x))) 0))
P42,1 & RW23      -->  * RW43:   All x,y:Any + =(*(y x) -(*(y -(x))))
RW24,1 & RW43,1   -->  * P53:    All x,y:Any + =(*(y -(x)) -(*(y x)))
A7,1 & A10,1      -->  * P235:   All x,y,z:Any - =(+(*(z y) *(x y)) 0)  + =(+(z x) 0)  + =(y 0)
A8,1 & P235,1     -->  * P236:   All x,y:Any - =(+(y *(x y)) 0)  + =(+(1 x) 0)  + =(y 0)
A2,1 & P236,1     -->  * P237:   All x,y,z:Any - =(+(z +(y *(x +(z y)))) 0)
                                              + =(+(1 x) 0)
                                              + =(+(z y) 0)
P237,1 & A6       -->  * RW238:  All x,y,z:Any - =(+(z +(y +(*(x z) *(x y)))) 0)
                                              + =(+(1 x) 0)
                                              + =(+(z y) 0)
A9,1 & RW238,1    -->  * P239:   All x,y:Any - =(+(1 +(y +(x *(x y)))) 0)
                                            + =(+(1 x) 0)
                                            + =(+(1 y) 0)
RW2,1 & P239,1    -->  * P240:   All x:Any - =(+(1 *(x -(x))) 0)  + =(+(1 x) 0)  + =(+(1 -(x)) 0)
P240,1 & P53      -->  * RW241:  All x:Any - =(+(1 -(*(x x))) 0)  + =(+(1 x) 0)  + =(+(1 -(x)) 0)
```

```
RW26,1 & RW241,1  --> * P242:   - =(+(1 -(1)) 0)  + =(+(1 c_1) 0)  + =(+(1 -(c_1)) 0)
P242,1 & P28,1    --> * R243:   + =(+(1 c_1) 0)  + =(+(1 -(c_1)) 0)
R243,2 & A2,1     --> * P244:   All x:Any + =(+(0 x) +(1 +(-(c_1) x)))  + =(+(1 c_1) 0)
P244,1 & A3       --> * RW245:  All x:Any + =(x +(1 +(-(c_1) x)))  + =(+(1 c_1) 0)
P28,1 & RW245,1   --> * P246:   + =(-(-(c_1)) +(1 0))  + =(+(1 c_1) 0)
P246,1 & RW23     --> * RW247:  + =(-(-(c_1)) 1)  + =(+(1 c_1) 0)
RW247,1 & RW24    --> * RW248:  + =(c_1 1)  + =(+(1 c_1) 0)
RW248,1 & T12,1   --> * R249:   + =(+(1 c_1) 0)
R249,1 & RW2,1    --> * P250:   + =(c_1 +(-(1) 0))
P250,1 & RW23     --> * RW251:  + =(c_1 -(1))
RW251,1 & T13,1   --> * R252:   []

--------------------------------------------------------------------------------

  q.e.d.
```

The proofs found by the computer have nothing to do with the hand-made paramodulation proofs. They construct a solution in the opposite direction and so the usage of variable paramodulation steps is avoided.

6 Conclusion

This report was originally motivated by a programming exercise: we just wanted to implement the Knuth-Bendix method directly into the MKRP-system using the trick as described in section 4. It came to our own surprise, that the system now by far out performed all of its previous versions with regard to equality and showed results we did not expect to be so significant.

Our conclusion is that a general equality reasoning procedure without Knuth-Bendix completion is unthinkable. Even if a subset of equations can be directed and completed beforehand it is still very worthwhile to have the completion procedure around. Almost every interesting mathematical theory has a part consisting of directable unit equations, albeit not "complete", and the Knuth-Bendix algorithm is the most efficient way to derive new interesting equations of which some are needed for almost every proof in the theory. In addition with this hard restriction of equation application (directed and completion) many interesting problems can be solved. In its constraining effect the usage of the Knuth-Bendix procedure seems to be comparable to the Waltz-effect in contrary to the proposition of K. Bläsius [Blä86]. The Waltz-effect is exploited in Vision for deriving consistent possibilities to interpret the topology of objects in a picture [Ric83b, pages 351-358].

But this seems to be so only for the standard completion procedure with unit equations, not for the extensions to arbitrary clauses. Here a heuristic approach with a strong depth search component that is orientated at the PROLOG strategy may be more adequate [BG90]. The problems occurring when imbedding a method for handling conditional completion in a resolution based theorem prover seem to be comparable to these of an imbedding in one based on polynomials [Den88].

Structure in axioms and theorems should be considered whereever this is possible but "normal" mathematical theories seem to have no usable structure, but at least it is not yet detected.

Acknowledgement

I have to thank Jörg Denzinger, Manfred Kerber, Christoph Lingenfelder, Hans Jürgen Ohlbach, Jörg Siekmann, and Christoph Weidenbach for readings and corrections of drafts of this paper.

References

[BBB+84] Susanne Biundo, Karl-Hans Bläsius, Hans-Jürgen Bürckert, Norbert Eisinger, Alexander Herold, Thomas Käufl, Christoph Lingenfelder, Hans Jürgen Ohlbach, Manfred Schmidt-Schauß, Jörg H. Siekmann, and Christoph Walther. The Markgraf Karl Refutation Procedure. SEKI-MEMO MK-84-01, Fachbereich Informatik, Universität Kaiserslautern, Institut für Informatik I, Universität Karlsruhe, Postfach 3049, D-6750 Kaiserslautern, Postfach 6380, D-7500 Karlsruhe 1, 1984.

[BDP87] Leo Bachmair, Nachum Dershowitz, and David Plaisted. Completion without failure. *CREAC, Lakaway, Texas*, 1987.

[BG85] M. Bidoit and M. C. Glaudel. PLUSS: Proposition pour un langage de spécification structurée. *Bicre + Globèle 45*, 1985.

[BG90] Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. *To appear on 10th CADE*, 1990.

[Blä86] Karl Hans Bläsius. *Equality Reasoning Based on Graphs.* PhD thesis, Universität Kaiserslautern, 1986.

[Bru75] M. Bruynooghe. The inheritance of links in a connection graph. Report CW2, Applied Mathematics and Programming Division, Katholieke Universiteit Leuven, 1975.

[Buc85] Bruno Buchberger. History and basic features of the critical-pair/completion approach. *Dijon*, 1985.

[Bun83] Alan Bundy. The computer modelling of mathematical reasoning. *Academic Press, London*, 1983.

[CD85] R. J. Cunningham and A. J. J. Dick. Rewrite systems on a lattice of types. *Acta Informatica*, 22:149–169, 1985.

[Dar68] J. L. Darlington. Automatic theorem proving with equality substitution and mathematical induction. *Machine Intelligence*, 3:113–127, 1968.

[Den88] Jörg Denzinger. EQTHEOPOGLES Ein Theorembeweiser für die Prädikatenlogik erster Stufe – basierend auf Rewrite Techniken. Diplomarbeit, Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern, 1988.

[Der87] Nachum Dershowitz. Rewriting systems. Draft, 1987.

[Dic85] A. J. J. Dick. Eril – equational reasoning, an interactive laboratory. In *Proceedings Eurocal Conference, Linz*, 1985.

[Dig79] V. J. Digricoli. Resolution by unification and equality. In *Proceedings 4th Workshop on Automated Deduction*, Austin, 1979.

[EFT78] H.-D. Ebbinghaus, J. Flum, and W. Thomas. Einführung in die mathematische logik. *Darmstadt*, 1978.

[Eis89] Norbert Eisinger. *Completeness, Confluence, and Related Properties of Clause Graph Resolution.* PhD thesis, Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern, 1989.

[EOP89] Norbert Eisinger, Hans Jürgen Ohlbach, and Axel Präcklein. Elimination of redundancies in clause sets and clause graphs. SEKI Report SR-89-10, Fachbereich Informatik, Universität Kaiserslautern, D-6750 Kaiserslautern, October 1989.

[Fag83] F. Fages. Formes canoniques dans les algèbres booléennes et application à la démonstration automatique en logique de premier ordre. *Thèse de 3ème Cycle, Paris*, 1983.

[FGJM85] K. Futatsugi, Joseph A. Goguen, J. P. Jouannaud, and José Meseguer. Principles of OBJ2. In *Proceedings 12th ACM Symposium on Principles of Programming Languages*, 1985.

[GI88] Jean H. Gallier and T. Isakowitz. Rewriting in order-sorted equational logic. Draft, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA 19104-6389, USA, 1988.

[GJM85] Joseph A. Goguen, J.-P. Jouannaud, and José Meseguer. Operational semantics of order-sorted algebra. In *Proceedings 12th ICALP, LNCS*, pages 221–231. Springer, 1985.

[GM85] Joseph A. Goguen and José Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In Doug DeGroot and Gary Lindstrom, editors, *Functional and Logic Programming*, pages 295–363. Prentice Hall, Englewood Cliffs, NewJersey 07632, 1985.

[GS86] Jean H. Gallier and Wayne Snyder. A general complete E-unification procedure. *Philadelphia*, 1986.

[GS89] Jean H. Gallier and Wayne Snyder. Complete sets of transformations for general E-unification. Report MS-CIS-89-12, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA 19104-6389, USA, December 1989.

[Her30] J. Herbrand. Récherches sur la théorie de la démonstration. *Travaux de la societé des sciences et de lettre de Varsovie, Class III Science mathématique et physique*, 33, 1930.

[HO80] G. Huet and D. Oppen. Equations and rewrite rules: a survey. *Technical Report CSL-III, SRI International*, 1980.

[HR86] J. Hsiang and M. Rusinowitch. A new method for establishing refutational completeness in theorem proving. In *Proceedings 8th CADE, LNCS*, Oxford, 1986. Springer.

[Hul80] J. M. Hullot. Canonical forms and unification. In *Proceedings 5th Workshop on Automated Deduction*, pages 318–334, 1980.

[Hut89] D. Hutter. Induction. *Talk at Deduktionstreffen*, 1989.

[JKK83] J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proceedings ICALP*, pages 361–373, 1983.

[JL87] J.-P. Jouannaud and P. Lescanne. Rewriting systems. *Technology and Science of Informatics*, 6(3):181–199, 1987.

[JW86] J.-P. Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In *Proceedings 3rd IFIP Conference on Formal Description of Programming Concepts*, Lyngby, 1986.

[Kap84] S. Kaplan. Fair conditional term rewriting systems: Unification, termination and confluence. *Laboratoire de Recherche en Informatique, Université d'Orsay*, 1984.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[Kir85] C. Kirchner. Méthodes et outils de conception systematique d'algorithmes d'unification dans les théories equationelles. *Thèse d'état, Université de Nancy I*, 1985.

[Kir87] C. Kirchner. Methods and tools for equational unification. *Colloquium on Resolution of Equations in Algebraic Structures*, 1987.

[KK89] C. Kirchner and H. Kirchner. Constrained equational rewriting. *Third International Workshop on Unification, UNIF*, 1989.

[Kow75] R. Kowalski. A proof procedure using connection graphs. *JACM*, 22(4), 1975.

[KZ89] D. Kapur and H. Zhang. A case study of the completion procedure: Proving ring commutativity problems. *State University of New York at Albany*, 1989.

[LO84] E. Lusk and R. Overbeek. A short problem set for testing systems that include equality reasoning. *Argonne National Laboratory*, 1984.

[Men87] E. Mendelson. *Introduction to Mathematical Logic.* The Wadsworth & Brooks/Cole Mathematics Series. Wadsworth, 3rd edition, 1987.

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

[MMR86] A. Martelli, C. Moiso, and G. F. Rossi. Lazy unification algorithms for canonical rewrite systems. 1986.

[Mor69] J. B. Morris. E-resolution: An extension of resolution to include the equality relation. In *Proceedings 1st IJCAI*, pages 287–294, 1969.

[Ohl87] Hans Jürgen Ohlbach. Link inheritance in abstract clause graphs. *Journal of Automated Reasoning*, 3(1):1–34, 1987.

[Ohl89] Hans Jürgen Ohlbach. Abstraction tree indexing for terms. In Hans-Jürgen Bürckert and Werner Nutt, editors, *UNIF'89 Extended Abstracts of the 3rd Int. Workshop on Unification*, pages 131–136, 1989.

[OL80] R. Overbeek and E. Lusk. Data structures and control architecture for implemention of theorem-proving programs. In *Proceedings 5th CADE*, 1980.

[OS89] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgarf Karl Refutation Procedure. SEKI-REPORT SR-89-19, Fachbereich Informatik, Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern, 1989.

[Pet83] Gerald E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM (Society for Industrial and Applied Mathematics) Journal of Computing*, 12(1):82–100, February 1983.

[Prä85] Axel Präcklein. Ein Reduktionsmodul für einen automatischen Beweiser. Diplomarbeit, Universität Karlsruhe, Postfach 3049, D-6750 Kaiserslautern, März 1985.

[Ric83a] M. Rice. The construction of a complete minimal set of contextual normal forms. In *Proceedings 1st Eurocal*, London, 1983.

[Ric83b] Elaine Rich. *Artificial Intelligence.* International Student Edition. McGraw Hill Book Company, 1983.

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

[Rus87] M. Rusinowitch. Démonstration automatique par des téchniques de reécriture. Thèse de Doctorat d'État en Mathématique, Nancy, 1987.

[RW69] G. Robinson and L. Wos. Paramodulation and theorem proving in first order theories with equality. *Machine Intelligence*, 4, 1969.

[Sib69] E. E. Sibert. A machine-oriented logic incorporating the equality axioms. *Machine Intelligence*, 4:103–133, 1969.

[Sie75] Jörg H. Siekmann. Stringunification. Memo CSM-7of, Essex University, 1975.

[Sie88] Jörg H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 1988.

[SNMG87] Gert Smolka, Werner Nutt, José Meseguer, and Joseph A. Goguen. Order-sorted equational computation. In *Proceedings CREAS Workshop*, 1987.

[Spe84] Volker Sperschneider. Logik. Script for Lectures at the Universität Karlsruhe, 1984.

[SS87] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. SEKI-Report SR-87-16, Fachbereich Informatik, Universität Kaiserslautern, December 1987.

[SS88] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* PhD thesis, Universität Kaiserslautern, 1988.

[Sti84] M. E. Stickel. A case study of theorem proving by the Knuth-Bendix method discovering that $x^3 = x$ implies ring commutativity. In *Proceedings 7th CADE*, 1984.

[Sti85] M. E. Stickel. Automated deduction by theory resolution. *JAR*, 1(4):333–357, 1985.

[SW80] Jörg H. Siekmann and Graham Wrightson. Paramodulated connection graphs. *Acta Informatica*, 13:67–86, 1980.

[WOLB84] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning Introduction and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1984.

[WRCS67] L. Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem prooving. *JACM*, 14:698–706, 1967.

[YS86] J. H. You and P. A. Subramanyou. E-unification algorithms for a class of confluent term rewriting systems. In *Proceedings 13th ICALP*, 1986.

[ZK88] H. Zhang and D. Kapur. First order theorem proving using conditional rewrite rules. In Ewing Lusk and Ross Overbeek, editors, *Proceedings 9th CADE*, pages 1–20, Argonne, Illinois, USA, 1988. Springer.

[ZR85] H. Zhang and J. L. Rémy. Contextual rewriting. In *Proceedings 1st Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science*, pages 46–62, Berlin, 1985. Springer.