Completion and Equational Theorem Proving using Taxonomic Constraints

Jörg Denzinger
Department of Computer Science
University of Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
E-mail: denzinge@informatik.uni-kl.de

Abstract

We present an approach to prove several theorems in slightly different axiom systems simultaneously. We represent the different problems as a taxonomy, i.e. a tree in which each node inherits all knowledge of its predecessors, and solve the problems using inference steps on rules and equations with simple constraints, i.e. words identifying nodes in the taxonomy. We demonstrate that a substantial gain can be achieved by using taxonomic constraints, not only by avoiding the repetition of inference steps in the different problems but also by achieving run times that are much shorter than the accumulated run times when proving each problem separately.

1 Introduction

The problem we are interested in is equational theorem proving (although taxonomic constraints can also be used by other theorem proving approaches that are based on the generation of facts): Given a set E of equations and a goal $s = t$, we want to prove $s =_E t$. Methods based on the Knuth-Bendix completion (see [KB70]) have turned out to be highly efficient for this problem. Very sophisticated heuristics have been developed to search goal directed in the search space (see [DF94]). Also, distributed theorem provers have been designed and implemented (see [AD93], [ADF95]) to enhance the power of these provers.

In this paper we focus on the problem to prove several theorems $s_i = t_i$ from (slightly) different axiom systems E^i, i=1,...,n, simultaneously. Clearly, if all the E^i are unrelated, each of the theorems has to be proved separately. But in many situations the axiom systems have large parts in common. Such a situation occurs, for example, when one wants to prove several conditional consequences of a mathematical structure. Then one may want not to repeat the same inference steps for different proofs. Here we propose to organize the axiom systems as a *taxonomy* T, a tree in which with each node a set of facts is associated, initially axioms, that are inherited by all successors of the node and a set of theorems to prove. Such taxonomies play also an important role in KL-ONE based knowledge representation formalisms (see [BS95]).

The inference rules of our system $\mathcal{TAX}$ reflect the structure of the taxonomy T by initially associating with each axiom a taxonomic constraint w describing the path from the root of T to the node N that contains the axiom. In a similar way taxonomic constraints are added to the theorems to prove. Inferences between axioms and/or theorems are only allowed if their nodes are on a common path. This can be tested easily by determining whether one of the corresponding constraints is a prefix of the other one. The result of an inference step is stored in the deeper one of the two nodes involved.

Our concept of a constraint is quite different from the common concept of constraints used in deduction processes (see [Bü90] or [KKR90]) that stem from logic programming (see, for example [Sm89]). Normally, constraints are added to some already derived fact F to describe the validity of F. In other words, a constraint C assigned to a fact F (a clause, an equation, a formula) is a logical formula. Then σ(F) is valid for those substitutions σ only that satisfy C. In this context the problem arises how to handle the constraints, especially when and how to solve them. These problems do not appear in our context.

As we will show by examples, our approach allows one to simultaneously solve sets of problems with less inference steps. Since the constraint handling is extremely simple this results directly in faster proofs.

This report is organized as follows: After this introduction, we will provide the necessary definitions to formulate our method in section 2. In section 3 we present an inference system for unfailing completion with taxonomic constraints, deal with the completeness of this system and provide control algorithms for the system we found useful. In section 4 we present experiments with our method. In section 5 we discuss some situations in which completion with taxonomic constraints is very useful and in section 6 we give a conclusion and utter a few remarks about future work.

2 Basic Definitions

Equational theorem proving deals with solving the following problem:

Input: E, a set of equations over a fixed signature sig;
$s = t$, a goal equation over sig
Question: Does $s = t$ hold in every model of E ?

In this paper we will deal with the following modification of this problem:

Input: $\{E^i | 1 \leq \text{i} \leq \text{k}\}$, all E^i sets of equations over a fixed signature sig;
k goal equations $s_i = t_i$ over sig
Question: Does for all i $s_i = t_i$ hold in every model of E^i ?

Let us first take a closer look at the first problem. Let $Th(E)$ denote the set of equations over sig that hold in every model of E. By Birkhoff's theorem we have $s = t \in Th(E)$ iff

s can be transformed into t by *replacing equals by equals*. Provers based on rewriting and completion techniques developed by Knuth and Bendix ([KB70]), improved to unfailing completion ([HR87], [BDP89]), have proven to be quite successful in solving this first problem.

We assume the reader to be familiar with rewriting and completion techniques. An overview is given in [AM90] or [DJ90].

A signature $sig = (S, F, \tau)$ consists of a set S of sorts, a set F of operators and a function $\tau : F \rightarrow S^+$ that fixes the input and output sorts of the operators. Let $\mathcal{T}(F, V)$ denote the set of terms over F and a set V of variables. By t/p we denote the subterm of t at position p and by t[p$\leftarrow$s] the replacement of t/p in t by s. By $\mathcal{T}(F) = \mathcal{T}(F, \emptyset)$ we denote a set of *ground terms* over F. Let K be a set of new constants. A *reduction ordering* $\succ$ is a well-founded ordering on $\mathcal{T}(F \cup K, V)$ that is compatible with substitutions and the term structure, i.e. $t_1 \succ t_2$ implies $\sigma(t_1) \succ \sigma(t_2)$ and $t[p \leftarrow t_1] \succ t[p \leftarrow t_2]$. If $\succ$ is total on $\mathcal{T}(F \cup K)$ then $\succ$ is called a *ground reduction ordering*.

A *rule* is an oriented equation, written as l $\rightarrow$ r, such that $Var(r) \subseteq Var(l)$. A set R of rules is *compatible* with $\succ$ if $l \succ r$ for every $l \rightarrow r$ in R. If E is a set of equations then $R_E = \{\sigma(u) \rightarrow \sigma(v) \mid u \doteq v \text{ in } E,\ \sigma \text{ a substitution},\ \sigma(u) \succ \sigma(v)\}$ is the set of orientable instances of equations in E. (We use $u \doteq v$ to denote $u = v$ or $v = u$.) Finally, we have $R(E) = R \cup R_E$. If l $\rightarrow$ r $\in R(E)$ and $\sigma(l) \equiv t/p$ and $t[p \leftarrow \sigma(r)] \equiv s$, then t is called *reducible* to s and we write t $\Rightarrow$ s. If a term t is not reducible with any element of $R(E)$, then it is in *normal form*.

Let $u \doteq v$ and $s \doteq t$ be equations in $E \cup R$. Let u/p be a non-variable subterm of u that is unifiable with s, say with most general unifier $\sigma = mgu(u/p, s)$. Then $\sigma(u)[p \leftarrow \sigma(t)] = \sigma(v)$ is in $Th(R \cup E)$. If $\sigma(u)[p \leftarrow \sigma(t)]) \not\succ \sigma(u)$ and $\sigma(v) \not\succ \sigma(u)$, then $\sigma(u)[p \leftarrow \sigma(t)] = \sigma(v)$ is a *critical pair* of R, E.

Since we will base our inference system for unfailing completion with taxonomic constraints on the inference system $\mathcal{U}$ presented in [AD93], which is a slight improvement of the system presented in [BDP89], we need also another ordering on terms, namely the encompassment ordering $\rhd$. It is the strict part of the quasi-ordering $\trianglerighteq$ defined by $s \trianglerighteq t$ iff $\sigma(t) \equiv s/p$ for some substitution σ and some position p.

In general, a proof procedure for equational theorem proving using unfailing completion works as follows: Input is the set E and g, a ground equation over $F \cup K$ (the skolemized goal $s = t$) and a ground reduction ordering $\succ$. The procedure uses sets R,E and CP. The input equations are put into CP (therefore E = $\emptyset$). Then the following loop is repeated until the normal forms of the terms of g are the same or subsumed by an equation of E or until the set CP is empty: Select an equation s'=t' out of CP. Let s" and t" be normal forms of s' and t' with respect to R(E). If neither s" $\equiv$ t" nor s"=t" is subsumed by an equation in E, then all critical pairs between s"=t" and E and R are added to CP (resp. their normal forms). If s" and t" are comparable with respect to $\succ$, then the appropriate rule is added to R, else s"=t" is added to E. All elements of E and R that can be reduced with the new rule or equation are removed from R or E and their normal forms are added to CP.

In order to solve the second problem given above we will use taxonomic constraints. Therefore we have to define what a taxonomy is.

Definition 2.1 (Taxonomy)

A tree T in which each node N consists of a set E_N of equations, a set R_N of rules and a set G_N of goal equations over a fixed signature *sig* is called a *taxonomy*. If a node N has n successors $N_1,...,N_n$, then these successors are ordered and the i-th node in this ordering is assigned the number i. Therefore each node N can be described by the path from the root to it, represented by a word $\mathrm{w_N}$ in $\mathbb{N}_+^*$: $\mathrm{w_N} = \epsilon$ if N is the root and $\mathrm{w_N} = \mathrm{w_M}\mathrm{i}$, if N is the i-th successor of node M.

The accumulated sets of equations E^N and rules R^N are defined recursively. If N is the root, then $E^N = E_N$ and $R^N = R_N$. If N is a successor of node M, then $E^N = E_N \cup E^M$, $R^N = R_N \cup R^M$.

The equality relation $=_N$ to a node N of T is defined as $=_N \, = \, =_{E^N \cup R^N}$.

Note that for our purposes it is sufficient to define a taxonomy as a tree. Some readers may find it useful to use directed acyclic graphs, as can be the case in some taxonomy definitions that are used in knowledge representation.

3 Unfailing Completion and Theorem Proving with Taxonomic Constraints

In this section we will present our unfailing completion with taxonomic constraints by means of an inference rule system called $\mathcal{TAX}$. As we will see, correctness and completeness of $\mathcal{TAX}$ follow directly from correctness and completeness of standard unfailing completion as presented in [BDP89]. However, some thoughts about fair selection strategies for constraint critical pairs are necessary. We will provide a fair selection strategy that does not have to take the constraints into account and give insight into some other implementational aspects. Finally, we will discuss obvious variations of $\mathcal{TAX}$ and the possible effects of these variations.

3.1 The inference system

Many experiences have shown that for theoretical reasons it is useful to present calculi in general and completion algorithms in particular in form of inference systems. Then, rather than having to prove for each small variation of an algorithm completeness again, one defines so-called fairness criteria that guarantee that each control strategy to the inference rules that fulfills these criteria produces a complete algorithm. We will also use this method for presenting our constraint completion method. Before we present our inference system, we have to define the form of the taxonomic constraints and their semantics, we have to transfer our set of problems into a taxonomy and we have to transfer the taxonomy back into starting sets for our completion inference system.

Definition 3.1 (taxonomic constraint, constraint equation/rule/goal)
Let T be a taxonomy and N a node of T. The word w_N to N is called the *taxonomic constraint* to N.
If w is a taxonomic constraint, s=t an equation, $l \rightarrow r$ a rule and u=v a goal equation, then s=t|w is a *constraint equation*, $l \rightarrow r|w$ a *constraint rule* and u=v|w a *constraint goal*.

As already stated, we want to solve several proof problems $s_i =_{E^i} t_i$ simultaneously by means of transforming them into a taxonomy and then performing completion with constraint rules, equations and goals obtained from the taxonomy. It is our goal to change standard unfailing completion as little as possible to accomplish this task. Since the constraints of the rules, equations and goals are sufficient to represent the taxonomy T from which they were obtained we will identify in the following a node N of T and its constraint w_N.

As a result of performing completion we have to expect that the sets E_N, R_N and G_N of a node N of a taxonomy change if an appropriate inference rule is applied. But we will prove that $=_N$ remains the same during the whole completion process for each node N of the taxonomy.

Standard unfailing completion works on sets of equations and rules. In order to solve our problem we will also need sets of goals. Initially, these sets have to be obtained from a taxonomy T. Before we will define our inference system we have to construct such a taxonomy out of our proof problems $s_i =_{E^i} t_i$. Since there are several ways to construct a taxonomy out of these problems this task has to be done by the user of our system $\mathcal{TAX}$. However, the following construction can be used to obtain an initial taxonomy T to the proof problems.

The root node N_0 of T is defined by
$$E_{N_0} = \bigcap_{i=1,\ldots,n} E^i,\ R_{N_0} = \emptyset \text{ and } G_{N_0} = \{s_i = t_i | E^i = E_{N_0}\}.$$
Then we partition the set $M = \{E^i | E^i \neq E_{N_0}\}$ into several sets M_1 to M_m, such that $M_i \cap M_j = \emptyset$ for all $i \neq j$, $\bigcup_{j=1,\ldots,m} M_j = M$ and $\bigcap_{E \in M_j} E \neq \emptyset$ for all j. For each M_j a node N_j as successor of the last node is constructed by
$$E_{N_j} = \bigcap_{E \in M_j} E \setminus E_{N_0},\ R_{N_j} = \emptyset \text{ and } G_{N_j} = \{s_i = t_i | E^i = E^{N_j}\}.$$
Then for each M_j this construction is repeated on and on until for all i = 1,...,n $s_i = t_i \in \bigcup_{N \in T_0} G_N$.

Note that there may be several partitions of a set M into M_1 to M_m and one of these partitions (preferably with small m and big sets E_{N_j}) has to be selected.

Definition 3.2 (Inference system $\mathcal{TAX}$)
Let T be a taxonomy. The inference system $\mathcal{TAX}$ works on triples (E,R,G), where E is a set of constraint equations, R a set of constraint rules and G a set of constraint goals. The initial set (E_0, R_0, G_0) to T is defined by
$$E_0 = \{s=t|w \mid s=t \in E_N \text{ for some } N \in T \text{ and w the constraint to } N\}$$
$$R_0 = \emptyset$$
$$G_0 = \{s=t|w \mid s=t \in G_N \text{ for some } N \in T \text{ and w the constraint to } N\}.$$

The inference rules of $\mathcal{TAX}$ are as follows:

1. Orient

$$\frac{E \cup \{s \doteq t|w\}, R, G}{E, R \cup \{s \rightarrow t|w\}, G} \quad \text{if } s > t.$$

2. Generate

$$\frac{E, R, G}{E \cup \{s \doteq t|w\}, R, G}$$

if $l_1 = r_1|w_1, l_2 = r_2|w_2 \in$ R $\cup$ E, s=t critical pair to $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ with mgu σ, $\sigma(l_1) \not< \sigma(r_1)$ and $\sigma(l_2) \not< \sigma(r_2)$, w_2=w_1u and w=w_2 or w_1=w_2u and w=w_1.

3. Simplify equation

$$\frac{E \cup \{s \doteq t|v\}, R, G}{E \cup \{u \doteq t|v\}, R, G} \quad \text{if } s \Rightarrow \text{u with } l \rightarrow r|w \in R(E),\ v = wv',\ s \rhd l.$$

4. Simplify right side of rule

$$\frac{E, R \cup \{l_1 \rightarrow r_1|w_1\}, G}{E, R \cup \{l_1 \rightarrow s|w_1\}, G} \quad \text{if } r_1 \Rightarrow s \text{ with } l_2 \rightarrow r_2|w_2 \in R(E),\ w_1 = w_2 v.$$

5. Simplify left side of rule

$$\frac{E, R \cup \{l_1 \rightarrow r_1|w_1\}, G}{E \cup \{s \doteq r_1|w_1\}, R, G}$$

if $l_1 \Rightarrow s$ with $l_2 \rightarrow r_2|w_2 \in R(E)$, $w_1 = w_2 v$, $l_1 \rhd l_2$.

6. Subsume equation

$$\frac{E \cup \{s_1 \doteq t_1|w_1, s_2 \doteq t_2|w_2\}, R, G}{E \cup \{s_2 \doteq t_2|w_2\}, R, G}$$

if $s_1/p \equiv \sigma(s_2)$, $t_1 \equiv s_1[p \leftarrow \sigma(t_2)]$, $s_1 \rhd s_2$, $w_1 = w_2 v$.

7. Simplify goal

$$\frac{E, R, G \cup \{s \doteq t|w\}}{E, R, G \cup \{u \doteq t|w\}} \quad \text{if } s \Rightarrow u \text{ with } l \rightarrow r|v \in R(E),\ \text{w} = \text{vv'}.$$

8. Subsume goal

$$\frac{E \cup \{s_1 \doteq t_1|w_1\}, R, G \cup \{s_2 \doteq t_2|w_2\}}{E \cup \{s_1 \doteq t_1|w_1\}, R, G}$$

if $s_2/p \equiv \sigma(s_1)$, $s_2[p \leftarrow \sigma(t_1)] \equiv t_2$, $w_2 = w_1 v$.

9. Delete goal

$$\frac{E, R, G \cup \{s \doteq t|w\}}{E, R, G} \quad \text{if s} \equiv \text{t}$$

10. Success

$$\frac{E, R, \{s \doteq t|w\}}{Success} \quad \text{if s} \equiv \text{t}$$

So, $\mathcal{TAX}$ deals with facts that are associated with paths in a taxonomy. For manipulating a fact of a given level of the path only facts with the same or higher level, representing more general knowledge, can be used. The results of inferences involving facts of different levels will always belong to the deeper, more specialized level.

Note that there are some differences between $\mathcal{TAX}$ and the very general method of constraint theorem proving described in [KKR90]. We will discuss these differences in section 3.4.

3.2 Theoretical aspects

Since the inference rules of $\mathcal{TAX}$ are essentially the same rules as those in [AD93], the necessary additional definitions and the proofs to establish correctness and completeness of [BDP89] can be taken over. Thus we have

Theorem 3.1 (correctness of $\mathcal{TAX}$)

Let (E,R,G) be the constraint equations, rules and goals to a taxonomy T. Let further (E',R',G') be the result of applying one of the rules of $\mathcal{TAX}$ to (E,R,G). Then for all nodes N of T $=_{E^N \cup R^N} = =_{E'^N \cup R'^N}$ holds.

Proof:

We reduce the problem to standard unfailing completion by considering a node N of T with constraint w_N. For rules 7 to 10 there is nothing to prove, because only the set G^N of N may be changed.

Since we allow only the simplification of term pairs by rules whose constraints are prefixes of the constraints of the term pairs (see also the comment at the end of section 3) and since critical pairs get the longer constraint, the node N and E^N and R^N are only affected, if both term pairs the inference rule uses have constraints that are prefixes of w_N. But this means, that these term pairs are in E^N and R^N. Then the correctness of the un-constraint inference system of [AD93] guarantees that $=_{E^N \cup R^N} = =_{E'^N \cup R'^N}$ holds. q.e.d.

An important property of an algorithm based on an inference rule system is a fair selection of applications of the inference rules. As already stated, for completion only a fair selection of critical pairs is required. Again, the definition of fairness of [BDP89] can be taken over to constraint critical pairs.

Definition 3.3 (Fairness)

A derivation $(E_i, R_i, G_i)_{i \in \mathbb{N}}$, where $(E_{i+1}, R_{i+1}, G_{i+1})$ is derived from (E_i, R_i, G_i) by application of a rule of $\mathcal{TAX}$, is called *fair*, if for each critical pair u = v|w to E^∞,R^∞ there is an i such that u =v|w $\in E_i$. Here, $E^\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$ and $R^\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$.

So, if we omit the constraint w, this would be the usual definition of fairness. Now we have

Theorem 3.2 (completeness of $\mathcal{TAX}$)

Let T be a taxonomy to the input problems $s_i =_{E^i} t_i$ and let $\succ$ be a ground reduction ordering and $(E_j, R_j, G_j)_{j \in \mathbb{N}}$ an derivation that is fair with E_0,R_0 and G_0 as defined before. Then for each pair of ground terms (s,t) with $s =_{E^i} t$ for an $i \in$ 1,...,n, there is an $j \in \mathbb{N}$ such that the normal forms of s and t with respect to $R_j(E_j)$ are identical.

Proof:

As in [BDP89] one can define a proof ordering $\succ_{\mathcal{U}}$ based on $\succ$ and show that for each proof for $s =_{E^i} t$ using elements of (E_j, R_j, G_j) with appropriate constraints that is no rewrite proof, there is a proof using elements of $(E_{j+1}, R_{j+1}, G_{j+1})$ (again with appropriate constraints) that is equal or smaller with respect to $\succ_{\mathcal{U}}$. In fact, one can use the same proof ordering as in [BDP89] and the same proof totally neglecting the constraints. They only restrict the rules and equations that can be used for a particular proof. But one has to make the following modification:
Since $\mathcal{TAX}$ does not allow the simplification of term pairs with constraint w using elements of $R(E)$ with an constraint wv, v $\neq \epsilon$, there are fewer simplifications possible than without constraints. For the proof of $s =_{E^i} t$ in (E_j, R_j, G_j) this means, that according to [BDP89] one could apply a simplification inference that is not possible with respect to $\mathcal{TAX}$. But, since a match is always also an mgu, there is a critical pair, i.e. a generation step, that can be applied to do the trick, resulting in a smaller proof.
The fairness guarantees that eventually to each proof that is no rewrite proof there will be a (E_j, R_j, G_j) containing a smaller one. Therefore for each pair of ground terms (s,t) with $s =_{E^i} t$ a j_0 will be reached such that the proof to $s =_{E^i} t$ in $(E_{j_0}, R_{j_0}, G_{j_0})$ is a rewrite proof, since $\succ_{\mathcal{U}}$ is Noetherian. q.e.d.

3.3 Practical aspects

The problem we have to face is how to guarantee derivations to be fair. An obvious solution is to use the FIFO-strategy for selecting the critical pairs, but it is well known that FIFO performs very badly. Therefore more intelligent strategies are needed. One way would be to extend a known (intelligent) strategy for unfailing completion to deal also with the constraints. Fortunately, this is not necessary. A very often used selection strategy, the smallest-component strategy of [Hu80] (we call it AddWeight, see [AD93]), produces fair derivations also for $\mathcal{TAX}$ without the need of considering the constraints.

AddWeight always selects a critical pair with the smallest term weight, which (in our system) is computed by counting a function symbol as 2 and a variable as 1. The term weight of each term of a critical pair is added to get the weight of a pair (hence the name *Add*Weight). Since there is only a limited number of term pairs that have a lower term weight than a given critical pair and since there is also only a limited number of term pairs with the same weight, AddWeight is a fair selection strategy with respect to the fairness defined in [BDP89]. Since the number of nodes in T is also limited, each term pair can only occur with a limited number of different constraints. Therefore there is only a limited number of term pairs with constraints that have a lower weight than a given constraint critical pair and this means that AddWeight is indeed fair.

Another aspect when looking for a selection strategy (or heuristic, if one is not interested in completeness) for critical pairs that is often critized is the absence of any goal orientation. The same criticism may also be directed towards our system $\mathcal{TAX}$. But, as in the case of simple unfailing completion, goal oriented selection heuristics, as defined in [DF94], can be used. If we do not store the critical pairs in one list sorted according to one selection strategy, but use for each one of the nodes of T a seperate list, then the selection from these lists can be organized according to different selection strategies. Since there are several goals to solve, a goal oriented heuristic should only be used for those critical pair lists that contain one goal only. The other lists should employ a strategy as, for example, AddWeight. Although fairness is of no concern when goal orientation is involved, the selection of the critical pair list from which the next critical pair is chosen should be fair, i.e. each list is chosen regularly. But, so far our experiments have shown that there is no need for such complex selection mechanisms; AddWeight was always sufficient (see section 4).

As we have seen, from a theoretical point of view only the sets E_i and R_i are interesting. Nevertheless, we included the sets G_i of goals in $\mathcal{TAX}$ and even several inference rules mainly dealing with goals, because the goals are very important for implementations of $\mathcal{TAX}$. An empty list G_i does not only indicate that all initial problems are solved and the run can be terminated, but the absence of any goals in a part of T (from a node down to all leaves of that part of T) allows us also to stop choosing any critical pairs with constraints referring to this part of T.

3.4 Discussion of $\mathcal{TAX}$

In this section we will discuss some technical differences between $\mathcal{TAX}$ and other constraint completion systems, as for example [KKR90]. Besides the main conceptual difference –constraints used in $\mathcal{TAX}$ define different contexts, a kind of different worlds deriving from each other, while in other systems constraints limit the allowed instances of a fact– there are some further differences that can be observed on the level of inference rules.

First, we do not have an inference rule that merges constraints and do also not provide a function to do a merging. It is possible, that the same rule, equation or goal (up to renaming of variables) occurs with two different constraints. Then this rule, equation or goal has to be represented twice (with respect to $\mathcal{TAX}$), one time for each constraint. We think that the designer of a completion algorithm to $\mathcal{TAX}$ should decide, whether he wants to use sets of constraints, thus avoiding double representation of term pairs, or wants to allow only one constraint, thus avoiding any splitting of constraint term pairs, which is, for example, the result when a term pair can be reduced with respect to one constraint but can not be reduced with respect to another one. Splitting means here, that the constraint of the term pair is splitted into a term pair with constraints that allow the reduction and into another one with all other constraints. So, the duplication of term pairs is introduced again.

To avoid splitting in simplification inference rules we also did not allow the simplification of a term pair whose constraint is a prefix of the constraint of the element of $R(E)$ that would reduce the unconstraint pair. If one wants to allow such simplifications one has to introduce a second list of constraints, a so-called outlist, and any application of an inference rule is only possible, if there is no element w in outlist, that is a prefix of a constraint of the other term pair used in the inference rule. Again, this results not only in an inference system that is more difficult to understand, but also in more difficulties when implementing a completion procedure. We will also see that using $\mathcal{TAX}$ as given in our definition results in a decrease of inference steps made and in decreasing run times with respect to proving each problem seperately. Therefore, it seems not necessary to us to complicate $\mathcal{TAX}$ any more.

Finally, we do not have a concept for "solving" constraints. Since we do not collect constraints that restrict substitutions, it is not necessary to do any computation regarding their combination and the solvability of them. This is one of the major problems of other constraint completion approaches, that so far is not satisfactory solved for many applications. It is of no problem at all for $\mathcal{TAX}$.

4 Experimental Results

Equational theorem proving with taxonomic constraints is intended for situations in which from several slightly different sets of equations several goals have to be proved. Typically, such situations occur when one wants to prove the validity of several conditional equations in a theory consisting of unconditional equations.

For our experiments we have chosen two such theories, namely *lattice ordered groups* and an equational axiomatization of the *propositional calculus*, and for each theory we have chosen several conditional equations of which we generated several problem sets we solved using our implementation of $\mathcal{TAX}$. We will compare run times, numbers of rules and equations generated, numbers of critical pairs computed and numbers of reductions made of our implementation using constraints and the cumulated results obtained with our implementation when proving each conditional equation alone. All experiments were made on a SUN Sparc 20 and the times are given in seconds. Note that our implementation is in C (based on the DISCOUNT system, see [ADF95]), but does not use indexing techniques or realizing lists of critical pairs as heaps.

4.1 Lattice ordered groups

The theory of lattice ordered groups is given by the following set of equations:

f(f(x,y),z) = f(x,f(y,z))	f(1,x) = x	f(i(x),x) = 1
l(l(x,y),z) = l(x,l(y,z))	l(x,y) = l(y,x)	l(x,x) = x
u(u(x,y),z) = u(x,u(y,z))	u(x,y) = u(y,x)	u(x,x) = x
f(x,l(y,z)) = l(f(x,y),f(x,z))	u(x,l(x,y)) = x	f(l(x,y),z) = l(f(x,z),f(y,z))
f(x,u(y,z)) = u(f(x,y),f(x,z))	l(x,u(x,y)) = x	f(u(x,y),z) = u(f(x,z),f(y,z))

In literature, as for example [KK74], lattice ordered groups are characterized as the combination of two mathematical structures, namely lattices and groups. Above can be seen the group operator f of arity 2, its neutral element 1 and the inverse operator i. A lattice is based on a partial ordering $\leq$ and two binary functions l and u, the greatest **l**ower bound and the least **u**pper bound of two elements. The two functions l and u can be used to get rid of the partial ordering $\leq$ with the help of the definition

x $\leq$ y iff l(x,y) = x or x $\leq$ y iff u(x,y) = y.

that was already used in the axiomatization above. Since there are two ways for eliminating $\leq$, there are several possible formulations to a given problem which is in the following indicated by the last letter of the names of the examples. So, lat2a und lat2b are two different formulations for the same problem. We present the conditional theorems already skolemized and in the form "additional axioms $\Rightarrow$ goal to prove".

ax_mono1a:	u(a,b) = b	$\Rightarrow$	u(f(a,c),f(b,c)) = f(b,c)
ax_mono1b:	l(a,b) = a	$\Rightarrow$	l(f(a,c),f(b,c)) = f(a,c)
ax_mono1c:	u(a,b) = b	$\Rightarrow$	l(f(a,c),f(b,c)) = f(a,c)
ax_mono2a:	u(a,b) = b	$\Rightarrow$	u(f(c,a),f(c,b)) = f(c,b)
ax_mono2b:	l(a,b) = a	$\Rightarrow$	l(f(c,a),f(c,b)) = f(c,a)
ax_mono2c:	l(a,b) = a	$\Rightarrow$	u(f(c,a),f(c,b)) = f(c,b)
lat1a:	u(a,1) = a	$\Rightarrow$	u(a,f(a,a)) = f(a,a)
lat2a:	u(a,1) = a, u(b,1) = b	$\Rightarrow$	u(a,f(a,b)) = f(a,b)
lat2b:	l(a,1) = 1, l(b,1) = 1	$\Rightarrow$	l(a,f(a,b)) = a
lat3a:	u(a,1) = a, u(b,1) = b	$\Rightarrow$	u(a,f(b,a)) = f(b,a)
lat3b:	l(a,1) = 1, l(b,1) = 1	$\Rightarrow$	l(a,f(b,a)) = a
p04a:	u(1,a) = a, u(1,b) = b	$\Rightarrow$	u(1,f(a,b)) = f(a,b)
p04b:	l(1,a) = 1, l(1,b) = 1	$\Rightarrow$	l(1,f(a,b)) = 1
p04c:	u(1,a) = a, u(1,b) = b	$\Rightarrow$	l(1,f(a,b)) = 1
p04d:	l(1,a) = 1, l(1,b) = 1	$\Rightarrow$	u(1,f(a,b)) = f(a,b)
p05a:	u(1,a) = 1, u(1,i(a)) = 1	$\Rightarrow$	1 = a
p05b:	l(1,a) = 1, l(1,i(a)) = 1	$\Rightarrow$	1 = a
p39a:	u(a,b) = a	$\Rightarrow$	u(i(a),i(b)) = i(b)
p39b:	l(a,b) = b	$\Rightarrow$	l(i(a),i(b)) = i(a)
p39c:	u(a,b) = a	$\Rightarrow$	l(i(a),i(b)) = i(a)
p39d:	l(a,b) = b	$\Rightarrow$	u(i(a),i(b)) = i(b)

Using a LPO with precedence i > f > n > u > 1 > a > b > c we get the results reported in Table 1 when proving each example alone. In addition to the rules for each example the same 8 unorientable equations were produced.

Ex.	**Run Time**	**Rules**	**crit. Pairs**	**Reductions**
ax_mono1a	21.59	118	8806	13610
ax_mono1b	21.99	122	9037	13976
ax_mono1c	22.50	122	9037	13976
ax_mono2a	21.30	108	8270	12819
ax_mono2b	21.48	114	8584	13282
ax_mono2c	21.17	108	8270	12819
lat1a	2.88	61	2230	2570
lat2a	4.00	90	3484	3921
lat2b	4.42	91	3521	3954
lat3a	27.56	293	20125	22079
lat3b	29.04	294	20204	22178
p04a	15.46	212	12338	12869
p04b	4.14	96	3670	4115
p04c	4.24	96	3670	4115
p04d	16.53	212	12338	12869
p05a	3.44	75	2819	3276
p05b	3.53	75	2816	3236
p39a	20.32	112	8492	13137
p39b	20.20	113	8535	13192
p39c	21.00	113	8535	13192
p39d	21.59	112	8492	13137

Table 1 : Lattice ordered groups: statistics for runs of one example only

We generated out of these examples the following experiments using taxonomic constraints.

LOGExp1 :
Examples included: all
Taxonomy :
Partition at first level: {ax_mono1a,ax_mono1c,ax_mono2a}, {ax_mono1b, ax_mono2b,ax_mono2c}, {lat1a,lat2a,lat3a}, {lat2b,lat3b}, {p04a,p04c}, {p04b, p04d}, {p05a}, {p05b}, {p39a,p39c}, {p39b,p39d}
Second level: {lat2a}, {lat3a}
Comment :
Obvious partition when proving all examples.

LOGExp2 :
Examples included: ax_mono1a, ax_mono1b, ax_mono1c, ax_mono2a, ax_mono2b, ax_mono2c, lat1a, lat2a, lat2b, lat3a, lat3b, p04a, p04b, p04c, p04d, p05a, p05b
Taxonomy :
Partition at first level: {ax_mono1a,ax_mono1c,ax_mono2a}, {ax_mono1b,

ax_mono2b,ax_mono2c}, {lat1a,lat2a,lat3a}, {lat2b,lat3b}, {p04a,p04c}, {p04b, p04d}, {p05a}, {p05b}

Second level: {lat2a}, {lat3a}

Comment :

Removing from LOGExp1 some examples that worked positive for our implementation.

LOGExp3 :

Examples included: ax_mono1a, ax_mono1c, ax_mono2a, lat1a, lat2a, lat3a, p04a, p04c, p05a, p39a, p39c

Taxonomy :

Partition at first level: {ax_mono1a,ax_mono1c,ax_mono2a}, {lat1a,lat2a,lat3a}, {p04a,p04c}, {p05a}, {p39a,p39c}

Second level: {lat2a}, {lat3a}

Comment :

All those examples that used the u-translation for $\leq$ in the premisses. Using the obvious partition.

LOGExp4 :

Examples included: ax_mono1b, ax_mono2b, ax_mono2c, lat2b, lat3b, p04b, p04d, p05b, p39b, p39d

Taxonomy :

Partition at first level: {ax_mono1b,ax_mono2b,ax_mono2c}, {lat2b,lat3b}, {p04b,p04d}, {p05b}, {p39b,p39d}

Second level: unnecessary

Comment :

All those examples that used the l-translation for $\leq$ in the premisses. Using the obvious partition.

LOGExp5 :

Examples included: lat3a,lat3b

Taxonomy :

Partition at first level: {lat3a}, {lat3b}

Second level: unnecessary

Comment :

Testing, how different the proofs of the u- and l-translations are. Obviously, they are very different.

Exp.	**Run times**		**Rules**		**crit. Pairs**		**Reduktions**	
	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$
LOGExp1	328.38	157.99	2737	1114	173273	80201	228332	95217
LOGExp2	245.27	136.90	2287	1046	139219	73764	175674	85423
LOGExp3	164.29	71.66	1400	596	87806	42706	115564	51589
LOGExp4	164.09	71.29	1337	597	85467	42782	112768	51647
LOGExp5	56.60	54.89	587	539	40329	38689	44257	42477

Table 2: Lattice ordered groups: statistics of runs using taxonomic constraints

4.2 Propositional calculus

An equational axiomatization of the propositional calculus is given by

C(T,x) = x	C(C(p,C(q,r)),C(C(p,q),C(p,r))) = T
C(p,C(q,p)) = T	C(C(p,C(q,r)),C(q,C(p,r))) = T
C(N(N(p)),p) = T	C(C(p,q),C(N(q),N(p))) = T
C(p,N(N(p))) = T	

This axiomatization is inspired by [Ta56]. The function C represents the logical implication, N the negation and T true. We selected out of [KW76] (pages 181, 182) the following conditional equations that are presented already skolemized and in the form "additional axioms ⇒ goal to prove", again. The numbers are the numbers of [KW76] (the examples not appearing here either used additional junctors or were too easy, meaning that they were solved in under one second).

9:	C(A,B) = T, C(A,N(B)) = T	⇒	N(A) = T
26:	C(A,B) = T, C(B,D) = T	⇒	C(A,D) = T
27:	C(A,B) = T	⇒	C(C(B,D),C(A,D)) = T
28:	C(A,B) = T	⇒	C(C(D,A),C(D,B)) = T
29:	C(C(A,B),D) = T	⇒	C(A,C(B,D)) = T
36:	C(A,N(A)) = T	⇒	N(A) = T
37:	C(N(A),A) = T	⇒	A = T
39:	N(A) = T	⇒	C(A,B) = T
40:	A = T	⇒	C(N(A),B) = T
44:	C(N(B),N(A)) = T	⇒	C(A,B) = T
45:	C(A,N(B)) = T	⇒	C(B,N(A)) = T
46:	C(N(A),B) = T	⇒	C(N(B),A) = T
55:	C(A,B) = T, C(N(A),B) = T	⇒	B = T

Using a LPO with precedence C > N > A > B > D > T we get the results of Table 3 when proving these examples without taxonomic constaints. No unorientable equations were produced.

Out of these examples we generated the following experiments.

PCExp1 :

Examples included: all

Taxonomy :

Partition at first level: {9,26,27,28,55}, {29}, {36}, {37}, {39}, {40}, {44}, {45}, {46}

Second level: {9}, {26}, {55}

Comment :

One way to partition the examples. The aim was to get one set on the first level as big as possible. Therefore all other sets consist of only one element.

Ex.	Run Time	Rules	crit. Pairs	Reductions
9	8.12	142	15275	25469
26	3.70	103	7840	13281
27	8.98	149	18044	30769
28	2.90	84	6380	10895
29	7.60	119	14273	24623
36	6.00	110	11950	20123
37	4.49	100	9878	16161
39	4.24	96	9067	16032
40	2.76	74	6292	10936
44	10.61	148	20335	33656
45	11.80	154	21674	35662
46	4.20	85	7461	12851
55	7.35	147	15017	24876

Table 3: Propositional calculus: statistics for runs of one example only

PCExp2 :
Examples included: all
Taxonomy :
Partition at first level: {9,45}, {26,27,28}, {46,55}, {36}, {37}, {39}, {40}, {44}
Second level: {9}, {26}, {55}
Comment :
Goal of the partition was to get as many sets with more than one element in the first level partition as possible.

PCExp3 :
Examples included: 9,26,27,28,44,45
Taxonomy :
Partition at first level: {9,26,27,28}, {44}, {45}
Second level: {9}, {26}
Comment :
Part of biggest partition with two further examples.

PCExp4 :
Examples included: 9,26,27,28,44,45,55
Taxonomy :
Partition at first level: {9,26,27,28,55}, {44}, {45}
Second level: {9}, {26}, {55}
Comment :
Whole biggest partition with same two examples as PCExp3.

PCExp5 :
Examples included: 28,29,36,37,39,40,44,45,46
Taxonomy :
Partition at first level: {28}, {29}, {36}, {37}, {39}, {40}, {44}, {45}, {46}
Second level: unnecessary

Comment :
All examples that can be put together without having a second level.

PCExp6 :
Examples included: 29,36,37,39,40,44
Taxonomy :
    Partition at first level: {29}, {36}, {37}, {39}, {40}, {44}
    Second level:
Comment :
Subset of PCExp5.

PCExp7 :
Examples included: 9,26,27,28,45,46,55
Taxonomy :
    Partition at first level: {9,45}, {46,55}, {26,27,28}
    Second level: {9}, {55}, {26}
Comment :
Partition of PCExp2 without further one-element sets.

PCExp8 :
Examples included: 9,40,44,45,46,55
Taxonomy :
    Partition at first level: {9,45}, {46,55}, {40}, {44}
    Second level: {9}, {55}
Comment :
Two smaller sets that can be used as partition with additional examples.

Exp.	**Run times**		**Rules**		**crit. Pairs**		**Reduktions**	
	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$
PCExp1	82.75	36.31	1511	407	163086	54023	275334	89557
PCExp2	82.75	29.48	1511	419	163086	50867	275334	85054
PCExp3	46.11	23.81	780	267	89548	38443	149735	62570
PCExp4	53.46	23.67	927	300	104565	41756	174611	67913
PCExp5	54.60	21.58	970	279	106910	35225	180939	59227
PCExp6	35.70	14.44	647	227	71395	29485	121531	50197
PCExp7	47.05	21.09	864	306	91691	37679	153803	61885
PCExp8	44.84	21.72	750	282	86053	35149	143450	57616

Table 4: Propositional calculus: statistics of runs using taxonomic constraints

4.3 Discussion of the results

The first, obvious and most important result of our experiments is that not only the number of inferences done by $\mathcal{TAX}$ is substantially smaller than the accumulated sum of the runs of single examples, but also the run times (see Table 2 and Table 4). This shows that our claim that taxonomic constraints are easy and efficient to handle is true. But typically the ratio of $\sum$ (the accumulated sum) to $\mathcal{TAX}$ is for rules, critical pairs and reductions better, i.e. slightly higher, than for the run time.

If we compare the two example domains, the number of equations that are common in all examples is for lattice ordered groups twice the number of that for propositional calculus. Concerning the experiments in which all examples were used (LOGExp1, PCExp1, PCExp2) we have more examples to prove for lattice ordered groups than for propositional calculus. Therefore one would expect that the ratio of $\sum$ to $\mathcal{TAX}$ would be in all statistics much better for lattice ordered groups than for propositional calculus. But this doesn't hold true.

The reasons for this phenomenon are the usage of AddWeight and the structure of the two domains. The additional axioms for the examples of both domains are quite short. This means that they produce many critical pairs that are also short and therefore will be selected by AddWeight prior to many of the critical pairs between elements of the common axioms. So, in both domains the number of common axioms is not so important.

If we take a closer look at the examples of the domain lattice ordered groups we can observe that the additional axioms either use l or u (due to the two translations of $\leq$). Therefore some proofs do not have many steps in common, although the number of steps is quite similar (again AddWeight effects this concentration on differrent directions). This is illustrated by the examples lat3a and lat3b. LOGExp5 solves those examples together and it can be seen that $\mathcal{TAX}$ needs nearly the same number of rules, critical pairs and reductions as the accumulated sum of the single runs (and the same run time, of course). Therefore the examples of the domain lattice ordered groups have to be divided into two groups, those using the u-translation and those using the l-translation. If we look at the results of $\mathcal{TAX}$ when given only the examples of one group (experiments LOGExp3 and LOGExp4), then the ratio of $\sum$ to $\mathcal{TAX}$ is better in all statistics than for LOGExp1.

Another expectation one has is that the addition of more examples from a domain –that do not require totally different proofs– does not increase the statistics of $\mathcal{TAX}$ as much as the statistics of $\sum$. This is indeed the case as proven by experiments LOGExp2 and LOGExp1, PCExp3 and PCExp4, PCExp7 and PCExp2 or PCExp6 and PCExp5. For all these pairs of experiments the increase of $\mathcal{TAX}$ is less than that of $\sum$ for all statistics of the Tables 2 and 4.

Some of the experiments in the domain propositional calculus demonstrate the effects of different partitions of the examples (i.e. PCExp1, PCExp2, PCExp4 and PCExp8). In general one can say that our results do not allow to favor a certain partition heuristic. This is because all the examples of the problem, not only those that allow different partitions, decide the statistics of a run.

Another interesting question is the behaviour of $\mathcal{TAX}$ when proving examples that are totally different, i.e. with no equations in common. To answer this question we combined examples from both domains in single experiments. The results are given in Table 5. The experiment CombExp1 used one example from each domain, namely lat3b and 45. We selected them because they had the longest run times in their group. In experiment CombExp2 we added to the examples of LOGExp5 the examples of

PCExp3. In CombExp3 we solved the examples of LOGExp1 and PCExp2 together (which means that in CompExp3 all our examples were proved). In the experiments CombExp2 and CombExp3 the results of $\sum$ are the sum of the basic experiments, i.e. the sum of the results of LOGExp5 and PCExp3, resp. LOGExp1 and PCExp2.

Exp.	**Run times**		**Rules**		**crit. Pairs**		**Reduktions**	
	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$	$\sum$	$\mathcal{TAX}$
CombExp1	40.84	40.93	448	448	41878	41878	57840	57978
CombExp2	78.70	79.59	806	806	77132	77132	105047	105238
CombExp3	187.47	192.17	1533	1533	131068	131068	180271	180408

Table 5: Propositional calculus and lattice ordered groups combined

All of the three experiments show that the overhead produced by handling of the constraints is neglectable. The run times of the experiments are nearly the same as the sums. So, the gains provided by the constraints are not disturbed by the further examples.

Finally, there is the question whether our small restriction of interreduction (no reduction of term pairs that are higher up in the taxonomy than the rule to apply) causes any problems. This question can be negated. The ratio of $\sum$ to $\mathcal{TAX}$ for the number of rules or the number of critical pairs is in all our experiments comparable to this ratio for the number of reductions. It seems that the reductions mentioned above are not important for the performance of the system.

As conclusion of our discussion of the experiments one can say that the use of $\mathcal{TAX}$ results in a decrease with respect to both, number of inferences made and run time, compared to proving all examples alone. Since this decrease is quite substantial, because the overhead of using taxonomic constraints is neglectable, applications that need to prove several theorems in slightly different axiom systems should always use $\mathcal{TAX}$.

5 Dynamic taxonomies and some applications

So far, taxonomies did not change during completion. In the following we will discuss situations in which a certain change of the taxonomy used, namely a dynamic extension, is required. We have a so-called *dynamic taxonomy*, when during the proof task new successors to nodes of the taxonomy may be added. Of course, these new successors inherit the equations and rules of their ancestors. As long as the taxonomy remains finite, this dynamic change does not influence the correctness and completeness results given in this paper.

Note that the form of our constraints guarantees the necessary inheritance without any further actions to be taken in addition to simply adding the new equations and goals with the new constraints. Especially, there is no need for any copy actions.

In the following we will sketch two situations in which dynamic taxonomies in combination with $\mathcal{TAX}$ can be useful. In [KKO95], Kurihara, Kondo and Ohuchi presented a method to complete a given set of equations with respect to several reduction orderings. The intention of this method was, as in our case, to reduce the repetition of inference steps that is a result of trying the completion for each reduction ordering. The use of several reduction orderings was suggested to solve the problem that for some sets of equations the use of one reduction ordering may produce an infinite computation while another one results in a finite, convergent system. Unfortunately, before trying, one does not know which reduction ordering is the best one.

The method presented in [KKO95] has one drawback, namely that all reduction orderings must be given before the completion is started. So, to be on the safe side, one has, for example, to use all permutations of the precedence ordering to a LPO or RPO, although many comparisions between function symbols may never be necessary due to the form of the equations. So, dynamic extension is missing in this approach. But this dynamic extension can be achieved by using completion with taxonomic constraints. The nodes of our, now dynamic, taxonomy represent reduction orderings (or better, rules that were oriented according to these orderings) and the ordering of a successor node N to a node M is an extension of the ordering of M.

Another situation stems from using completion theorem provers in interactive proof environments that are used as proof assistants. Such an environment is, for example, the ILF-System [Da+94] in which the DISCOUNT-System is used for solving pure equational problems. Very often an user of such a system is interested in checking the consequences of some manipulations of parts of the axioms for proving a goal. This means that there is a given basis of equations to which another set of equations is added. Some of the equations of this second set may be withdrawn later and other equations may be added instead. Without the use of a dynamic taxonomy and $\mathcal{TAX}$ each change of the second set forces the user to start a new run of the completion prover that has to repeat all those inferences between the basis equations and the remaining equations of the second set that have already been made in prior experiments. This redundant work is very frustrating for the user.

If the user is able to order the equations of the second set with respect to the possibility that they are withdrawn, then he can use $\mathcal{TAX}$ with a degenerated taxonomy that has only one path in which each additional equation of the second set is assigned a new node according to the given order. In case the user wants to withdraw an equation and add some other ones instead a new node (or several new ones if some of the new equations may be withdrawn, again) is added as successor of the father of the node representing the withdrawn equation. So, assuming a careful planning of the experiments, many results of prior experiments can be used for new ones, thus improving the acceptance of the whole proof assistant.

6 Conclusion and Future Work

We have presented taxonomic constraints as a way to reduce the repetition of many inferences when one wants to prove theorems in slightly different axiom systems for the case of equational deduction by unfailing completion. By transforming all examples one wants to prove into a taxonomy and then transforming this taxonomy back into sets of constraint equations, rules and goals we were able to develop a theorem prover that did not only much less inference steps than those that were needed to prove each of the examples alone but also in less time. We also sketched several situations in which a theorem prover using taxonomic constraints is very useful.

Future work should center on exploiting the use of dynamic taxonomies, development of more selection heuristics for constraint critical pairs and the implementation and experimental evaluation of the idea presented in section 3.3 to use goal oriented selection heuristics. Also the use of taxonomic constraints in other theorem provers is of interest.

References

[AD93] **Avenhaus, J. ; Denzinger, J.**: *Distributing equational theorem proving*, Proc. 5th RTA, Montreal, LNCS 690, 1993, pp. 62-76; also available as SEKI-Report SR-93-06, University of Kaiserslautern, 1993.

[ADF95] **Avenhaus, J. ; Denzinger, J. ; Fuchs, M.**: *DISCOUNT: A system for distributed equational deduction*, Proc. 6th RTA, Kaiserslautern, LNCS 914, 1995, pp. 397-402.

[AM90] **Avenhaus, J. ; Madlener, K.**: *Term Rewriting and Equational Reasoning*, in R.B. Banerji (ed): Formal Techniques in Artificial Intelligence, Elsevier, 1990, pp. 1-43.

[BDP89] **Bachmair, L. ; Dershowitz, N. ; Plaisted, D.A.**: *Completion without Failure*, Coll. on the Resolution of Equations in Algebraic Structures, Austin (1987), Academic Press, 1989.

[BS85] **Brachman, R.J.; Schmolze, J.G.**: *On Overview of the KL-ONE Knowledge Representation System*, Cognitive Science 9(2), 1985, pp. 171-216.

[Bü90] **Bürckert, H.-J.**: *A Resolution Principle for Clauses with Constraints*, Proc. 10th CADE, Kaiserslautern, Springer, LNAI 449, 1990, pp. 178-192.

[Da+94] **Dahn, B.I. ; Gehne, J. ; Honigmann, T. ; Walther, L. ; Wolf, A.**: *Integrating Logical Functions with ILF*, Internal report, Institut für Reine Mathematik, Humbold-University, Berlin, 1994.

[DF94] **Denzinger, J. ; Fuchs, M.**: *Goal oriented equational theorem proving using teamwork*, Proc. 18th KI-94, Saarbrücken, LNAI 861, 1994, pp. 343-354; also available as SEKI-Report SR-94-04, University of Kaiserslautern, 1994.

[DJ90] **Dershowitz, N. ; Jouannaud, J.P.**: *Rewriting systems*, in J. van Leeuwen (Ed.): Handbook of theoretical computer science, Vol. B., Elsevier, 1990, pp. 241-320.

[HR87] **Hsiang, J. ; Rusinowitch, M.**: *On word problems in equational theories*, Proc. 14th ICALP, Karlsruhe, LNCS 267, 1987, pp. 54-71.

[Hu80] **Huet, G.**: *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems*, Journal of ACM, Vol. 27, No. 4, 1980, pp. 798-821.

[KB70] **Knuth, D.E. ; Bendix, P.B.**: *Simple Word Problems in Universal Algebra,* Computational Algebra, J. Leech, Pergamon Press, 1970, pp. 263-297.

[KK74] **Kokorin, A.I. ; Kopytov, V.M.**: *Fully ordered groups*, Halsted Press, 1974.

[KKO95] **Kurihara, M. ; Kondo, H. ; Ohuchi, A.**: *Completion for Multiple Reduction Orderings*, Proc. 6th RTA, Kaiserslautern, LNCS 914, 1995, pp. 71-85.

[KKR90] **Kirchner, C. ; Kirchner, H. ; Rusinowitch, M.**: *Deduction with symbolic constraints*, Revue d'Intelligence Artificielle 4(3), 1990, pp. 9-52.

[KW76] **Kleinknecht, R. ; Wüst, E.**: *Lehrbuch der elementaren Logik, Bd. 1: Aussagenlogik*, DTV-Verlag, 1976.

[Sm89] **Smolka, G.**: *Logic Programming over Polymorphically Order-Sorted Types*, Ph.D. thesis, University of Kaiserslautern, 1989.