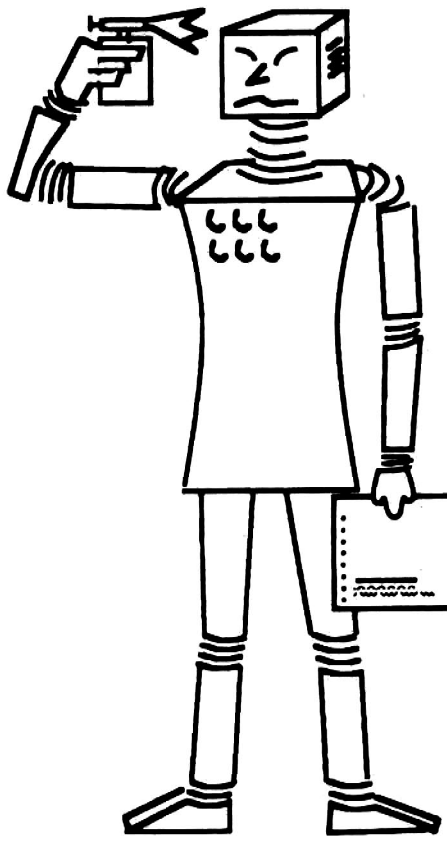

Learning proof heuristics by adapting parameters

Matthias Fuchs

SEKI Report SR-95-02

Learning proof heuristics by adapting parameters

Matthias Fuchs

SEKI Report SR-95-02

Learning proof heuristics by adapting parameters[*]

Matthias Fuchs
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: fuchs@informatik.uni-kl.de

March 10, 1995

Abstract

We present a method for learning heuristics employed by an automated prover to control its inference machine. The hub of the method is the adaptation of the parameters of a heuristic. Adaptation is accomplished by a genetic algorithm. The necessary guidance during the learning process is provided by a proof problem and a proof of it found in the past. The objective of learning consists in finding a parameter configuration that avoids redundant effort w.r.t. this problem and the particular proof of it. A heuristic learned (adapted) this way can then be applied profitably when searching for a proof of a similar problem. So, our method can be used to train a proof heuristic for a class of similar problems.

A number of experiments (with an automated prover for purely equational logic) show that adapted heuristics are not only able to speed up enormously the search for the proof learned during adaptation. They also reduce redundancies in the search for proofs of similar theorems. This not only results in finding proofs faster, but also enables the prover to prove theorems it could not handle before.

1 Introduction

Automated deduction is—at its lowest level—a search problem that spans huge search spaces. It is well known that automated deduction can for this reason be a very expensive task in terms of computing time and usage of computer memory when it comes to proving "challenging" theorems. Despite its superior inference rate the computer is in these cases mostly inferior to (human) mathematicians. One prominent reason for

[*]This work was supported by the *Deutsche Forschungsgemeinschaft (DFG).*

this fact is the ability of humans to apply information gained during past attempts to find (similar) proofs. Automated provers that lack this ability (and most of them do) tackle each problem as a complete novice. A substantial amount of redundant work is done by exploring regions of the enormous, in general infinite search space which do not contribute anything useful to finding a proof. Such regions could possibly be detected if the automated prover were able to fall back on past experience gathered while trying to solve a "similar" proof problem.

But striking problems arise when trying to reuse past experience in the context of automated deduction. First of all, unlike in other fields of research (such as planning or case-based reasoning, e.g., [Ca86], [Bu89]), the use of analogy in the wider sense in connection with deduction is much more difficult and hazardous (cp. [KN93]), because it is *not* the case that *"small changes of the problem description (usually) cause small changes of the solution"*. Moreover, even if the proofs of two problems are (intuitively) non-trivially similar[1], this similarity can very seldom be reduced to simple, easy to define symbolic mappings. Furthermore, the general undecidability of problems in the area of (automated) deduction adds to these tremendous difficulties.

Despite these rather slim prospects of success we still think that it is worthwhile trying to equip automated provers with the option to make use of past experience. The central idea of the method we are going to propose (see also [Sm83]) is the adaptation ("*tuning*") of parameters of a heuristic, which the automated prover at hand employs for controlling its inference machine. The aim of adaptation is to "tune" a heuristic so that it allows the automated prover using it to prove a given proof problem $\mathcal{A}'$ with "as little redundancy as possible". A proof $\mathcal{P}$ of $\mathcal{A}'$ found in the past is used to guide this learning process which is accomplished by a genetic algorithm. An adapted heuristic obtained this way can then be employed when proving a similar, novel problem $\mathcal{A}$. The reduction of redundancies w.r.t. the original problem $\mathcal{A}'$ and its proof $\mathcal{P}$, i.e., avoiding paths of the search space which do not (immediately) lead to $\mathcal{P}$ ("pruning" the search space), carries over to an improved performance w.r.t. a similar problem $\mathcal{A}$. Inherent in this method is a suitable compromise between exactness and flexibility which is necessary to exploit non-trivial similarities profitably.

The experiments conducted with our method for learning[2] heuristics in the field of equational reasoning are very promising. We were not only able to achieve outstanding speed-ups in several cases, but also enabled the prover to handle problems it could not cope with before, because its memory was swamped with redundant information. We want to emphasize that the problems in question include tasks such respected and powerful deduction systems as OTTER 3.0 ([Mc94]) and Herky ([Zh92]) have difficulties with (see subsection 6.4).

In the following sections details will be presented. First of all, section 2 will discuss general aspects of utilizing proofs found in the past and will introduce the fundamentals of our approach. Since a genetic algorithm will play an essential role for our learning

[1] "Trivial similarity" or "trivial relatedness" denote identity modulo renaming.

[2] We use the notions "adaptation" and "learning" as well as the expressions "learning (adapting) a heuristic" and "learning (adapting) the parameters (of a heuristic)" synonymously.

procedure, section 3 will give an overview of this topic. In section 4 the basics of the automated prover used for our experiments are described. In section 5 details of the central problematic of our method will be discussed. The results of our experiments are summarized in section 6, followed by a short discussion of related work in section 7. Finally, a summary in section 8 brings this report to a close.

2 Fundamentals

Utilizing proofs found in the past generally confronts us with the following situation: We are given a problem description $\mathcal{A} = (Ax, Th)$, and the task is to find a proof for the theorem Th based on the axioms in Ax. In order to accomplish this we make use of a proof found in the past. Let $\mathcal{A}' = (Ax', Th')$ be a problem for which a proof is known. $\mathcal{A}'$ is referred to as the *source*, while $\mathcal{A}$ is called the *target*.

The first idea that comes to mind when thinking about reusing a proof found in the past (the source proof) to prove a given theorem is to try to infer facts that are in some way similar to those needed for the source proof ("*derivational analogy*", e.g. [Ca86]). This method surely works perfectly when we have total correspondence between the source and the target. Then "similar" means "identical" and the target proof can be found (almost) without redundant work by merely re-enacting the source proof. But this case is not very exciting since we can achieve the same effect simply by storing and retrieving proofs from a database (*memory-based reasoning*). Things tend to become more interesting if source and target are "similar to some degree", but do not agree completely. In that case a meaningful similarity measure is much harder to define. Furthermore, in case the chain of reasoning induced by the source proof is disrupted, patching strategies must be available to compensate for the loss of guidance from the source proof. Such strategies are difficult to construct since the cause for a failure of the source proof will usually not be obvious. So, we are mostly left with bits and pieces.

The general disadvantage of such an approach is the fact that it stays too close to the source proof. This results in a significant inflexibility necessitating elaborate methods to cushion failures (i.e., deviations from the source proof). For this very reason we have chosen a different approach: Almost any automated prover disposes of a set of heuristics for controlling the application of its inference rules. Usually, these heuristics are parameterized. So, why don't we try to adapt (learn) these parameters in order to find the source proof more quickly? Of course by doing so the intention is not to improve overall performance of the respective heuristic, but to tailor a heuristic which is especially suitable for proofs like the source proof it was adapted to. In other words, in adapting existing heuristics we intend to capture ("learn") essential properties of a certain type of proof. An adapted heuristic should then be profitably applicable to proofs that share these properties "to some extent". This way of proceeding takes off our back the heavy load of designing patching strategies, since even an adapted heuristic should still be flexible enough to cope with minor deviations. This property of commonly used heuristics stems from their limitedness that will in general prevent any learning procedure from finding parameter configurations that allow for searching a proof totally without redundancy, i.e., without taking unnecessary steps. But this inherent inexactness is a benefit, because it is vital for the flexibility of an adapted heuristic, which in turn is crucial for its profitable applicability to non-trivially related problems. Furthermore, this approach avoids the necessity to pin down the (exact) nature of similarity between two related problems in order to be successfully applicable.

A further advantage of our approach is the non-existence of any kind of overhead *during application* on account of reusing proof experiences. All the work is done *once* during adaptation. As a matter of fact we actually reuse proofs *indirectly*: By adapting the parameters of a heuristic this heuristic "learns" the given (source) proof, being completely unaware of a (possible) future target. When the adapted heuristic is employed for tackling a similar problem the proof learned during adaptation is not consulted anymore. Its "essence" has been assimilated by the parameters of the heuristic in the course of adaptation.

We have now laid down the motivation for our approach, i.e., adapting parameters of heuristics. Hence it is time to think about how adaptation can be accomplished, which is (in general) a very demanding task. Commonly, a parameter of a heuristic is a flag or a numerical value, both of which can very conveniently be represented by one or several bits. Thus the complete set of parameters of a heuristic is representable as a string of bits. Since the basic structure of a heuristic remains untouched, any string of bits representing a parameter configuration is a (more or less suitable) solution of an adaptation. Hence the use of a genetic algorithm (GA) suggests itself. A GA is the core of our adaptive method. Its basics will therefore be outlined in the subsequent section, together with an explanation of its application to our problem.

3 Learning parameters using a genetic algorithm

3.1 Fundamentals of genetic algorithms

The genetic algorithm ([Ho75], [Da88], [Ra91])—GA for short—is an adaptive method based on principles known from general genetics and biological evolution. It is very useful for finding (near) optimal solutions to problems in many domains. It differs from other optimization techniques in that it maintains a set of individuals (usually with a fixed size) which is called a *population* or *generation*. Each of these individuals corresponds to a (sub-optimal) solution of the given problem. An individual is thus a representation of a solution that suits the GA. The GA constructs new individuals (and hence new solutions) using the best ones of its current population, replacing those considered least fit ("*survival of the fittest*"). The assessment of individuals is accomplished by the so-called *fitness function*. The construction of new individuals is achieved by applying *genetic operators* which basically reflect and simulate the processes involved in biological reproduction. The most important genetic operators are *crossover*, *mutation* (and *inversion*). They are defined by

Definition 3.1 ((One-point) crossover) *Given two individuals $a_1 \dots a_n$ and $b_1 \dots b_n$, where the a_i and b_j each represent one bit, randomly choose $k \in \{2, \dots, n\}$ and then construct two (new) individuals*

$$a_1 \dots a_{k-1} b_k \dots b_n \quad \textit{and} \quad b_1 \dots b_{k-1} a_k \dots a_n \, .$$

It is possible to discard randomly or deterministically one of the two descendants ("children") or to retain them both.

Definition 3.2 (Mutation) *Given an individual $c_1 \dots c_n$, each bit c_i may be inverted with a probability p_{mut}.*

Definition 3.3 (Inversion) *Given an individual $c_1 \dots c_n$, randomly choose $k_1, k_2 \in \{1, \dots, n\}$ with $k_2 > k_1$. Replace then the sub-string $c_{k_1} \dots c_{k_2}$ with $c_{k_2} \dots c_{k_1}$.*

An important prerequisite for the GA to be applicable is a structure of the individuals that is amenable to these genetic operators. Common is a representation as a string of bits of a fixed length n.

A further essential component of the GA is the fitness function. The fitness function actually establishes the (only) connection between the GA and the current problem. It assesses the fitness of a solution associated with an individual and hence performs among other things the transformation of a bit string into an actual description of a solution. Biologically speaking, the fitness function transforms the genotype (bit string) into the phenotype (actual solution) and provides the environment for testing the fitness of the respective "organism". But in usual applications of GAs that transformation process ("morphogenesis") is merely a translation of the genotype (i.e., a representational change) rather than a real development (e.g., a bit string of length $8n$ is transformed into a set of n 8-bit integers, for instance n numbers ranging from -128 to 127). The fitness rating is used to determine the chances of an individual to become reproductive (i.e., to be one of the "parents" in the crossover operation) or to be eliminated. Usually, p_{retain} percent of the individuals rated best by the fitness function are admitted to reproduction, their offspring replacing the remaining $100 - p_{retain}$ percent. Chances for becoming reproductive may be proportional to the fitness rating or some other function of it.

We can now give a concise description of the GA in algorithmic form:

Step 1: Initialize a population of size n_{pop} by (usually randomly) generating n_{pop} bit strings (individuals) of length n.
Initialize the "generation counter" $c_{gen} := 1$.

Step 2: Rate the individuals of the current population with the fitness function.

Step 3: Retain the p_{retain} percent best individuals. Replace the rest with offspring of the best ("survival of the fittest"). Offspring is produced by randomly selecting (with the probability possibly depending on the fitness rating) two individuals among those best ones and applying the crossover operator. The mutation and inversion operators are applied to the offspring, the latter operator with a probability p_{inv}. Increment the generation counter c_{gen} and continue with step 2.

Notes: The GA terminates if the best individual achieves a rating which is better than a given limit or if the number of generations c_{gen} exceeds a given threshold. Each run through the loop formed by step 2 and step 3 may also be called a *cycle* (of the GA).

We conclude this subsection with a few remarks on the implementation of a GA used for our experiments.

The best p_{retain} percent individuals have an equal chance to take part in reproduction. Both "children" are retained.

$$p_{retain} = 30\% \qquad p_{inv} = 10\%$$
$$n_{pop} = 50 \qquad p_{mut} = 10\%$$

It is worth noting that although we have a rather simple implementation of a GA (there are no additional features such as maintaining diversity through a variable mutation rate, for instance) and although the population size $n_{pop} = 50$ is quite small compared to common standards, our GA performed sufficiently well. Furthermore, we did (so far) not experiment intensively with the settings of the above parameters since we recognized that finding better solutions is mainly a question of finding a better fitness function. As a matter of fact, designing an appropriate fitness function is the crucial problem of our application as the following subsection will reveal.

3.2 Our application of the GA

The previous subsection has presented the major components of a GA, namely (the representation of) individuals, the genetic operators and the fitness function. In our case, individuals correspond to parameter configurations of a heuristic for selecting the next (deductive) inference to be performed by an automated prover. By restricting the range of each parameter a parameter configuration can be represented by a string of bits of fixed length which is constructed by concatenating the bit representation of each parameter of the configuration. Hence no problems are raised regarding the applicability of the genetic operators. So, the major problem that remains is the design of an appropriate fitness function. The fitness function is in general very important for any GA application, because it establishes the (only) connection between the (abstract) underlying search method and the actual (optimization) problem. Our problem consists in finding a parameter configuration for a heuristic so that an automated prover using this (adapted) heuristic can find a proof of a given proof problem $\mathcal{A} = (Ax, Th)$ with "minimal redundancy". The—as we shall see—*necessary* guidance during the search for an optimal parameter configuration is provided by a (source) proof $\mathcal{P}$ of $\mathcal{A}$ found in the past.

Let $\mathcal{W}$ be a generic heuristic, and ω an instance of $\mathcal{W}$, i.e., the parameters of $\mathcal{W}$ are set to specific values. The quality of an ω is expressed by the amount of redundancy

ω would produce if a proof of $\mathcal{A}$ guided by ω were attempted. So, the ideal and most exact way to assess an ω consists in proving or attempting to prove that Th follows from Ax using ω and subsequently analyzing statistical data collected during the search. But clearly enough this is impractical. The fitness function has to be applied to each (new) individual during each cycle[3], and hence a fitness function that needs at least a couple of seconds (which has to be considered the *lower* limit to obtain expressive statistical data during a proof attempt) for rating just one individual is untenable.

The following considerations offer an efficient alternative. Most redundancies during a search for $\mathcal{P}$ are caused by inferring facts which do not contribute to finding $\mathcal{P}$. The source proof $\mathcal{P}$ corresponds to a particular set of paths in the search space. With the help of $\mathcal{P}$ all (potentially) inferable facts can be classified into two categories: On the one hand, there are "useful" (*positive*) facts, which have to be inferred in order to follow such a path given by $\mathcal{P}$. On the other hand, "useless" (*negative*) facts represent facts that lead away from these paths, entailing redundant search effort. (We emphasize that the terms "positive" and "negative" must be seen in the context of finding the particular proof $\mathcal{P}$.) Consequently, an obvious method for estimating an ω consists in measuring its ability to distinguish positive facts from negative ones, i.e., its ability to cut off misleading paths. Here, we have to face another problem: While the set P of positive facts is finite (and usually rather small), the set of negative facts is in general infinite. Hence we have to confine ourselves to a finite subset N of the set of (all) negative facts. P and N can be extracted from the proof run that yielded $\mathcal{P}$, for instance. But there is one crucial problem in this approach, namely using a *static* N. Even in case we succeed in adapting $\mathcal{W}$ "perfectly", i.e., it associates weights[4] with the members of P and N so that the highest weight of a fact in P is still below the weight of any fact in N, we have absolutely no guarantee that such a perfect "adaptation" will carry over to an improved search for a proof. As a matter of fact, it is even possible for the adapted heuristic ω to perform more poorly than the heuristic originally employed to find $\mathcal{P}$. The reason for such a behavior is the high probability that ω may infer negative facts different from those in N, which might actually complicate the search for $\mathcal{P}$ even more. We therefore have to step back from the idea of using a *static* N.

The problems just outlined can—at least to a large extent—be compensated for by periodically updating N, i.e., by maintaining a *dynamic* N. The adaptive procedure (learning) then proceeds as follows: Starting with $N = \emptyset$, the automated prover is run for a given time T before calling the GA for the first time, and then each time the GA has executed n_c cycles. Each run through this outer loop (the inner loop corresponds to a cycle of the GA) will be referred to as an *iteration*. For these *update runs* the prover uses the ω currently rated best. If the prover actually succeeds in finding a proof, ω will be among the output of the adaptive procedure. Facts generated during time T which are not in P are considered as negative facts and are added to N. It must be emphasized that newly occurring (negative) facts are *added* to N and the "old" ones

[3]One cycle of the GA comprises the rating of all individuals, selecting the best and replacing the rest with offspring of the best.

[4]It is assumed that a heuristic associates a weight (a natural number) with all potentially inferable facts and that the fact with the smallest weight is actually inferred.

are *not* discarded. Otherwise, if replacing N each time, chances are that we might run into some form of "oscillation" where the same N appear cyclically, which may have severely disadvantageous effects on the GA's search for an optimum.

We have now explained the central role the GA plays regarding our learning task. We have also brought out that the central problematic of this GA application is the design of a fitness function. Through the discussion above it has become clear that—for efficiency and practicability reasons—we have to content ourselves with a fitness function that simulates rather than actually performs proof runs. This simulation is based on the sets P and N which depend on the source proof $\mathcal{P}$. We already pointed out that N has to be updated periodically. This measure is a remnant of the "ideal" fitness function which is indispensable to be able to achieve a "realistic" simulation. Section 5 will present the fitness function in a more detailed way. This necessitates that we can no longer refer to arbitrary provers. Therefore, and because we want to substantiate our considerations with experimental results (see section 6) we have to confine ourselves to a specific automated prover. We have chosen an equational prover based on the unfailing Knuth–Bendix completion procedure (*UKB-procedure*), namely the DISCOUNT system ([ADF95]). Our choice has mainly been favored by the existence of knowhow and implementations. We believe that our method can be utilized by almost any (automated) prover that uses parameterized heuristics to control its inference machine and explicitly infers facts.

Before describing the details of the fitness function in this specific context in section 5 we shall outline the foundations of the UKB-procedure in the following section in order to clarify the terminology and to provide some basic insight into its functioning as well as to present the heuristics that are candidates for adaptation.

4 Equational theorem proving with the UKB-procedure

In this section we shall describe the fundamentals of the unfailing Knuth–Bendix completion procedure ([KB70], [HR87], [BDP89]). The UKB-procedure can be used for solving the word problem in structures defined by equations in two ways. Firstly, it may attempt to *complete* the initial set of equations (the *axioms*) yielding—in case of success—a convergent (i.e., confluent and terminating) system which can then be used as a decision procedure for the word problem. Secondly, it may concentrate on a given word problem (the theorem) and stop as soon as it has been solved (semi-decision procedure). The UKB-procedure is based on the operational semantics of equational reasoning ("substituting terms by equal terms"). We shall now outline its foundations.

First, we have to define some basic notions.

Definition 4.1 (signature) *A* signature *is a tuple $sig = (\mathcal{F}, \tau)$ where $\mathcal{F}$ is a finite set of function symbols, and $\tau(f)$ denotes the arity of any $f \in \mathcal{F}$, i.e., $\tau(f) \in I\!N$. $f \in \mathcal{F}$ is called a* constant *if $\tau(f) = 0$.*

Definition 4.2 (term) *Let $\mathcal{V} = \{x, y, z, \ldots\}$ be a (enumerable) set of variables, $sig = (\mathcal{F}, \tau)$ a signature, $\mathcal{V} \cap \mathcal{F} = \emptyset$. The set of* terms *$Term(\mathcal{F}, \mathcal{V})$ is defined as follows. $t \in Term(\mathcal{F}, \mathcal{V})$ iff*
(1) $t \in \mathcal{V}$ or
(2) $t \equiv f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}$, $t_1, \ldots, t_n \in Term(\mathcal{F}, \mathcal{V})$ and $\tau(f) = n$.
If $\mathcal{V} = \emptyset$ then $Term(\mathcal{F}) := Term(\mathcal{F}, \emptyset)$ is the set of ground terms.

In the following, $\mathcal{V}$ will always denote a (enumerable) set of variables. Furthermore, $\mathcal{F}$ and τ will always refer to the signature $sig = (\mathcal{F}, \tau)$.

Definition 4.3 (substitution) *A function $\sigma : \mathcal{V} \longrightarrow Term(\mathcal{F}, \mathcal{V})$ is called a* substitution *with the (finite) domain $dom(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \not\equiv x\}$. σ is extended to $Term(\mathcal{F}, \mathcal{V})$ by $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ for all $f \in \mathcal{F}$, $\tau(f) = n$.*
Σ denotes the set of all substitutions.

Definition 4.4 (occurrences) *Let $t \in Term(\mathcal{F}, \mathcal{V})$. The set of occurrences in t, denoted $O(t)$, is defined by $O(t) = \{\varepsilon\}$, $t|_\varepsilon \equiv t$, if $t \in \mathcal{V}$ or t is a constant.*
Otherwise, if $t \equiv f(t_1, \ldots, t_n)$, $O(t) = \{ip \mid 1 \leq i \leq n,\, p \in O(t_i)\} \cup \{\varepsilon\}$, $t|_{ip} \equiv t_i|_p$, $t|_\varepsilon \equiv t$.
If $s \in Term(\mathcal{F}, \mathcal{V})$, $p \in O(t)$, then $t[p \leftarrow s]$ denotes the term obtained by replacing $t|_p$ with s.

Definition 4.5 (equation, specification) *A pair of terms $(t_1, t_2) \in Term(\mathcal{F}, \mathcal{V})^2$ is called an* equation, *usually written as $t_1 = t_2$. A set of equations E together with the corresponding signature sig is called a* specification *$spec = (sig, E)$.*

For equational reasoning, a problem $\mathcal{A} = (Ax, Th)$ as introduced in section 2 signifies that we have a specification $spec = (sig, Ax)$ and an equation Th which is the theorem to be proved (the *goal*). Throughout this report we assume that all variables occurring in (the equations of Ax or) Th are implicitly $\forall$-quantified (for extensions, i.e., $\exists$-quantified variables in Th, see [De93], pp. 19–41). In order to handle the variables properly in any case $Th \equiv s = t$ is negated and skolemized yielding $s' \neq t'$. In our case ($\forall$-quantification only), s' or t' are obtained from s or t, respectively, by merely replacing all variables with (new) Skolem constants. Skolemization also allows us to deal with theorems of the form $\forall \vec{x} : s_1 = t_1 \wedge \ldots \wedge s_n = t_n \rightarrow s = t$ (cp. section 6.3). Negation and skolemization produce $s'_1 = t'_1 \wedge \ldots \wedge s'_n = t'_n \wedge s' \neq t'$. Hence, $s'_1 = t'_1 \wedge \ldots \wedge s'_n = t'_n$ are added to the set of axioms, and $s' \neq t'$ is the actual goal. In the following we shall not rigorously distinct between the original theorem Th and its negated and skolemized counterpart, keeping in mind that the UKB-procedure works with the latter.

The general proceeding of the UKB-procedure when attempting to prove Th consists in inferring new equations from Ax (and its derived descendants). When possible the goal Th is rewritten using the equations derived so far. Rewriting a term t with an equation $s_1 = s_2$ or $s_2 = s_1$ means replacing a sub-term $t|_p$ of t ($p \in O(t)$), which is an instance of s_1, i.e., $t|_p \equiv \sigma(s_1)$ for some $\sigma \in \Sigma$, with $\sigma(s_2)$. Th is proved if both of its sides can be made identical this way.

A strong advantage of the UKB-procedure resides in the use of a reduction ordering $\succ$. Strictly speaking, equations are symmetrical, i.e., they can be applied from left to right or from right to left. If the sides of an equation can be compared with $\succ$ then this equation needs only be applied in the direction from the bigger side (w.r.t. $\succ$) to the smaller side. An equation $s_1 = s_2$ that can be compared with $\succ$ is called a *rule* and is written as $l \rightarrow r$, where $l = \max_{\succ}(\{s_1, s_2\})$ and $r = \min_{\succ}(\{s_1, s_2\})$. The definition of a reduction ordering follows.

Definition 4.6 (reduction ordering) *A partial ordering on $Term(\mathcal{F}, \mathcal{V})$ is a* reduction ordering *if it satisfies the following conditions (let $s_1, s_2, s, t \in Term(\mathcal{F}, \mathcal{V})$).*
(1) $s \succ t$ implies $\sigma(s) \succ \sigma(t)$ for all $\sigma \in \Sigma$ (compatibility with substitution)
(2) $s_1 \succ s_2$ and $p \in O(s)$ imply $s[p \leftarrow s_1] \succ s[p \leftarrow s_2]$ (compatibility with term structure)
(3) $\succ$ is well founded (there are no infinitely descending chains $s_1 \succ s_2 \succ \cdots$).
A ground reduction ordering *is a reduction ordering which is total on ground terms.*

The UKB-procedure can now be described in a more detailed way. As already sketched above new equations have to be inferred. This is accomplished by generating so-called critical pairs. For this process the reduction ordering can be employed as a restrictive means.

Definition 4.7 (critical pair) *Let $l_1 \sim_1 r_1$ and $l_2 \sim_2 r_2$ be rules or equations (i.e., $\sim_1, \sim_2 \in \{\rightarrow, =\}$), $\succ$ a reduction ordering, $p \in O(l_1)$, $\sigma \in \Sigma$ with $\sigma(l_1|_p) \equiv \sigma(l_2)$.*
$\langle u, v\rangle$ is a critical *pair of $l_1 \sim_1 r_1$ and $l_2 \sim_2 r_2$ if*

$$u \equiv \sigma(l_1)[p \leftarrow \sigma(r_2)] \quad , \quad v \equiv \sigma(r_1)$$

and

$$\sigma(r_1) \not\succ \sigma(l_1) \quad , \quad \sigma(r_2) \not\succ \sigma(l_2).$$

Note: *The latter two conditions are always true if the respective $\sim_i = \rightarrow$. Furthermore, $l_1 \sim_1 r_1$ and $l_2 \sim_2 r_2$ do not necessarily have to be distinct.*

The efficiency of the UKB-procedure depends largely on the application of *reduction* as the only rewriting inference. Reduction corresponds to a special case of rewriting, the restriction being that, if $\sigma(s_1) = \sigma(s_2)$ is the instance of the equation for rewriting by substituting $\sigma(s_2)$ for $\sigma(s_1)$, then $\sigma(s_1) \succ \sigma(s_2)$ must be satisfied. Hence, rules can always be used for reductions (in the direction of the arrow). For equations, since neither $s_1 \succ s_2$ nor $s_2 \succ s_1$, it must be tested if for the given instances $\sigma(s_1)$ and $\sigma(s_2)$ $\sigma(s_1) \succ \sigma(s_2)$ holds.

In order to solve the given problem $\mathcal{A} = (Ax, Th)$, i.e., to show $Ax \vdash Th$, the UKB-procedure operates on three sets R, E and CP. R is the set of rules, E the set of equations (their sides cannot be compared with the reduction ordering $\succ$) and CP is the set of critical pairs ("unprocessed equations"). Before we can present the UKB-procedure in an algorithmic form we still have to introduce the notions "normal form" and "interreduction".

A *normal form* of a term s is a term s' (also denoted $s\downarrow$) which can be obtained from s via reductions using the rules in R and the equations in E, and which cannot be reduced any further. In that case we say that $s\downarrow$ is in *normal form with reference to R and E*. Note that normal forms are not necessarily unique, i.e., there may exist more than one normal form of a term s. In case of termination (which is guaranteed here because of the reduction ordering) *at least* one normal form exists. Confluence entails the existence of *at most* one normal form. Hence convergence guarantees the existence of exactly one normal form. Normalization is a further advantageous concept of the UKB-procedure simplifying the available facts as much as possible, thus keeping them as concise as possible. An equation or rule is said to be normalized if both its sides are in normal form. Strongly related with normalization is *interreduction*. Interreducing R and E with a rule $l \rightarrow r$ $(u = v)$ means that it is attempted to reduce every rule of R and every equation of E with $l \rightarrow r$ $(u = v)$. In case of success the respective side of the rule[5] or equation in question is brought into normal form w.r.t. $R \cup \{l \rightarrow r\}$ and E (R and $E \cup \{u = v\}$). Of course, a rule or equation must not be reduced with itself.

Initially, $R = E = \emptyset$, $CP = Ax$ and the goal $G \equiv s' \neq t'$. The UKB-procedure then proceeds as follows:

```
while $CP \neq \emptyset$ and $s' \not\equiv t'$
do
  select a $\langle u', v' \rangle \in CP$
  $CP := CP \setminus \{\langle u', v' \rangle\}$
  $\langle u, v \rangle := \langle u', v' \rangle$ normalized w.r.t. $R$ and $E$
  if $u \not\equiv v$ then
    if $u \succ v$ or $v \succ u$ then
      let $l = \max_{\succ}(\{u, v\})$, $r = \min_{\succ}(\{u, v\})$
      interreduce $R$ and $E$ with $l \rightarrow r$
      $R := R \cup \{l \rightarrow r\}$
      normalize $G$ w.r.t. $R$ and $E$
      $CP_{new}$ := set of normalized critical pairs between $l \rightarrow r$ and $R$ and $E$
    else
      interreduce $R$ and $E$ with $u = v$
      $E := E \cup \{u = v\}$
      normalize $G$ w.r.t. $R$ and $E$
      $CP_{new}$ := set of normalized critical pairs between $u = v$ and $R$ and $E$
    fi
    $CP := CP \cup CP_{new}$
  fi
done
```

[5] A rule $l' \rightarrow r'$ in R whose left side could be reduced yielding l'' is moved from R to CP since $l'' \succ r'$ might no longer be valid.

Notes:

- If the above procedure stops with $CP = \emptyset$ and $s' \not\equiv t'$ then the theorem is not provable.
- The procedure may not stop at all (general undecidability of the problem '$Ax \vdash Th?$').
- This procedure may also be used for *completion* (see [KB70]) by omitting everything related to G.

The crucial, indeterministic step of the UKB-procedure is the selection of the next critical pair to become a rule or equation. A judicious choice can speed up proving (completion) considerably, whereas poor choices can slow it down extremely and even make it (practically) impossible. Due to the general undecidability of $Ax \vdash Th$ only heuristics can be applied to resolve this indeterminism.

A heuristic for selecting the next critical pair usually associates a weight (a natural number) with each critical pair. Commonly (and that is what we assume here) the critical pair with the lowest weight is the one considered the most suitable and will be selected. (If there are several critical pairs with the same lowest weight then the one that has been in the set CP for the longest time is picked (FIFO).)

The DISCOUNT system ([ADF95]) is an automated equational prover based on the procedure just described. It is implemented in C and features tools for proof analysis and extraction (see [DS94a], [DS94b]) we applied to extract source proofs. All our experimental results were obtained with this system. In the sequel we shall present the (generic) heuristics of DISCOUNT we used for the experiments documented in section 6.

The heuristics for choosing the next critical pair are based on a *weighting function* $\phi : \mathcal{F} \cup \mathcal{V} \to \mathbb{N}$, where $\phi(x) = w_v \in \mathbb{N}$ for all $x \in \mathcal{V}$ and $\phi(f) = w_f$ for all $f \in \mathcal{F}$. ϕ can be extended to $Term(\mathcal{F}, \mathcal{V})$ by defining

Definition 4.8 (weight of a term)

$$\phi(t) = \begin{cases} w_v, & \textit{if } t \equiv x \in \mathcal{V} \\ w_f + \sum_{i=1}^{n} \phi(t_i), & \textit{if } t \equiv f(t_1, \ldots, t_n),\ f \in \mathcal{F},\ n = \tau(f) \end{cases}$$

A generalized form ϕ_{lp} of ϕ is used by the heuristic "*linpol*". It assigns a linear polynom with the coefficients $c_1^f, \ldots, c_{\tau(f)}^f \in \mathbb{N}$ to each $f \in \mathcal{F}$.

Definition 4.9

$$\phi_{lp}(t) = \begin{cases} w_v, & \textit{if } t \equiv x \in \mathcal{V} \\ w_f + \sum_{i=1}^{n} c_i^f \cdot \phi_{lp}(t_i), & \textit{if } t \equiv f(t_1, \ldots, t_n),\ f \in \mathcal{F},\ n = \tau(f) \end{cases}$$

In the following let $\langle u, v \rangle$ be a critical pair, i.e., $u, v \in Term(\mathcal{F}, \mathcal{V})$.

Definition 4.10 *Let $\succ$ be the reduction ordering, $s \neq t$ the negated and skolemized goal (cp. section 4). Let furthermore for all $f \in \mathcal{F}$ $m_f \geq 1$ be factors expressing some structural difference between a critical pair $\langle u, v \rangle$ and $s \neq t$ (see appendix A for details). The heuristics add, max, gt, linpol and occnest are defined as follows:*

$$\begin{aligned}
add(\langle u,v\rangle) &= \phi(u) + \phi(v) \\
max(\langle u,v\rangle) &= \max(\{\phi(u), \phi(v)\}) \\
gt(\langle u,v\rangle) &= \begin{cases} \phi(u), & \textit{if } u \succ v \\ \phi(v), & \textit{if } v \succ u \\ \phi(u)+\phi(v), & \textit{otherwise} \end{cases} \\
linpol(\langle u,v\rangle) &= \phi_{lp}(u) + \phi_{lp}(v) \\
occnest_{s \neq t}(\langle u,v\rangle) &= (\phi(u)+\phi(v)) \cdot \prod_{f \in \mathcal{F}} m_f
\end{aligned}$$

(*Note*: *occnest* is a goal oriented heuristic and is therefore labeled with the current goal $s \neq t$. Cf. also appendix A.)

The parameters of these heuristics are the values w_v and w_f for all $f \in \mathcal{F}$, as well as the coefficients $c_1^f, \ldots, c_{\tau(f)}^f$ for all $f \in \mathcal{F}$ in case of *linpol*, and a flag in case of *occnest* (see appendix A).

The *default versions* of these heuristics (as used by DISCOUNT) are specified by the following choice of parameters: $w_v = 1$, $w_f = 2$ for all $f \in \mathcal{F}$, $\mathcal{D} = \mathcal{F}$ (only relevant for *occnest*, see appendix A), $c_i^f = 1$ for all $f \in \mathcal{F}$, $1 \leq i \leq \tau(f)$ (only relevant for *linpol*). The default version of *linpol* is hence equivalent to the default version of *add*. We shall therefore not distinguish between these two heuristics when considering default versions.

The following section will describe the details of the fitness function (cp. section 3) with respect to this particular equational prover. Once again we want to emphasize that our approach is not limited to this kind of provers, but can be applied to any kind of deductive system employing parameterized heuristics and explicitly inferring facts.

5 Designing a fitness function

As outlined in sections 2 and 3 our approach proceeds as follows: Given a proof $\mathcal{P}$ of a proof problem $\mathcal{A}' = (Ax', Th')$—the source proof—and a (generic) heuristic $\mathcal{W}$ (*add*, *max*, *gt*, *linpol* or *occnest*, cp. section 4) we are looking for values of the parameters w_v and w_f for all $f \in \mathcal{F}$ so that the respective instance ω of $\mathcal{W}$ enables the prover (in our case DISCOUNT) to find a proof of $Ax' \vdash Th'$ with as little redundancy as possible. If a future proof problem $\mathcal{A}$ is "similar" to $\mathcal{A}'$, then this particular ω is employed to guide the search. Thus, we indirectly reuse the proof $\mathcal{P}$ of $\mathcal{A}'$ by using ω which "learned" that proof.

The learning of the parameters of $\mathcal{W}$ is conducted by a genetic algorithm (GA). Representing solutions (i.e., parameter configurations resp. instances ω of $\mathcal{W}$) and constructing new ones by applying genetic operators is no problem at all (cp. section 3). The major difficulty is the design of an accurate, yet efficient fitness function. Section 3 explained its foundations, namely the use of the fixed set P of positive critical pairs (or facts in general) and a set N of negative critical pairs. P represents the steps necessary to attain the proof $\mathcal{P}$ of $\mathcal{A}'$ found in the past, whereas N stands for a subset of all inferences which do not contribute to this particular proof $\mathcal{P}$. We already discussed in section 3 why—in contrast to P—N cannot be fixed (static). Periodic proof runs have to update N in order to take into account the changing search behavior of the evolving instances ω of $\mathcal{W}$. In the sequel, we present technical details by gradually developing the fitness function.

5.1 First steps

So, the fitness function of the GA works with the two sets $P = \{\, \langle u_j, v_j \rangle \mid 1 \leq j \leq m \,\}$ and N, the latter being periodically updated after n_c cycles of the GA. In order to estimate the quality of an ω, i.e., its ability to prefer elements of P to elements of N, the following sets $\mathcal{I}_j^\omega(N) \subseteq N$ associated with each $\langle u_j, v_j \rangle \in P$ are pivotal. The elements of each $\mathcal{I}_j^\omega(N)$ are the *currently known* negative critical pairs which *may* be preferred to the respective $\langle u_j, v_j \rangle \in P$ during a search for $\mathcal{P}$ using ω.

$$\mathcal{I}_j^\omega(N) = \{\, \langle u, v \rangle \in N \mid \omega(\langle u, v \rangle) \leq \omega(\langle u_j, v_j \rangle) \,\}\,, \qquad 1 \leq j \leq m$$

The elements of $\mathcal{I}_j^\omega(N)$ are those negative critical pairs that are (potentially) preferred to $\langle u_j, v_j \rangle \in P$. It is quite obvious that ω should be considered the better the smaller the sets $\mathcal{I}_1^\omega(N), \ldots, \mathcal{I}_m^\omega(N)$ are. A "perfect" adaptation entails $\mathcal{I}_j^\omega(N) = \emptyset$ for all $1 \leq j \leq m$. Since this will hardly ever be the case we have to find a more subtle way of rating a given ω. So, the first idea for a fitness function ϑ_1 evaluating ω consists in adding up the sizes of $\mathcal{I}_1^\omega(N), \ldots, \mathcal{I}_m^\omega(N)$, i.e.,

$$\vartheta_1(\omega) = \sum_{j=1}^{m} |\mathcal{I}_j^\omega(N)|\,.$$

But this simple solution has many shortcomings mainly because of the following observation: It depends on two factors if the members of a $\mathcal{I}_j^\omega(N)$ are actually worked on before $\langle u_j, v_j \rangle$ is chosen when attempting a proof using ω. Firstly, a $\langle u, v \rangle \in \mathcal{I}_j^\omega(N)$ with $\omega(\langle u, v \rangle) = \omega(\langle u_j, v_j \rangle)$ is not given necessarily priority to. Secondly, so far nothing is said about the position of a $\langle u, v \rangle$ in the derivation graph w.r.t. ω; it might be the case that $\langle u, v \rangle$ has not yet been generated while $\langle u_j, v_j \rangle$ already is in the set CP. If this is the case, then $\langle u, v \rangle$ is not an obstacle for the selection of $\langle u_j, v_j \rangle$.

In this context we must point out that measuring a heuristic by its ability to prefer elements of P to elements of N can probably never be an equally good substitute for actually evaluating a proof or proof attempt. The main reason for this drawback is the set N. N is updated after each iteration by attempting a proof using one particular ω'—the best generated so far. N can consequently contain critical pairs which may be "irrelevant" for a ω other than ω'. Here, *irrelevant* critical pairs denote critical pairs in N which contribute to the assessment of ω in a possibly negative way by increasing the size of some of the $\mathcal{I}_j^\omega(N)$, but which would not have been selected before the respective $\langle u_j, v_j \rangle \in P$ if a proof using this ω had been attempted. Similarly, "relevant" critical pairs are critical pairs possibly not in N, but they would be members of some of the $\mathcal{I}_j^\omega(N)$ and they would have been selected if the UKB-procedure had been using ω. The following definition formalizes these notions.

Definition 5.1 (Relevance, Irrelevance) *Let ω be an instance of a heuristic $\mathcal{W}$ and $\langle u_j, v_j \rangle \in P$. A negative critical pair $\langle u, v \rangle$, i.e., $\langle u, v \rangle \notin P$, is* relevant *w.r.t. ω and $\langle u_j, v_j \rangle$ if $\omega(\langle u, v \rangle) \leq \omega(\langle u_j, v_j \rangle)$ and $\langle u, v \rangle$ would be selected before $\langle u_j, v_j \rangle$ if ω were used during a proof or proof attempt. $\langle u, v \rangle$ is* irrelevant *w.r.t. ω and $\langle u_j, v_j \rangle$ if $\omega(\langle u, v \rangle) \leq \omega(\langle u_j, v_j \rangle)$ and $\langle u, v \rangle$ would not be selected before $\langle u_j, v_j \rangle$ if ω were used during a proof or proof attempt.*

Note that relevance or irrelevance of a negative critical pair are to be seen w.r.t. a certain ω and a $\langle u_j, v_j \rangle \in P$, but not as a general property. A negative critical pair $\langle u, v \rangle$ may be relevant w.r.t. ω and $\langle u_j, v_j \rangle \in P$, but irrelevant w.r.t. ω' and $\langle u_k, v_k \rangle \in P$, or vice versa (where $\omega \not\equiv \omega'$ *or* $j \neq k$). Relevance or irrelevance of a critical pair for a given ω and a $\langle u_j, v_j \rangle \in P$ can only be decided if the UKB-procedure is started *each time a ω is to be evaluated*. This is exactly what we wanted to avoid. Therefore we have to find other ways to get around this problem. By employing the best ω during update runs we already diminish the number of irrelevant critical pairs. But this is achieved at the expense of also decreasing the number of relevant critical pairs. This dilemma of not having relevant critical pairs—which can make a "bad" ω look better than it actually is—and having irrelevant critical pairs—which can make a "good" ω look worse than it is—cannot be overcome completely unless computing the individual N of each ω to be rated. But this is—as we already discussed—too time consuming and hence impractical.

Before we take a look at improvements of ϑ_1 including measures taken to cope with the dilemma just described, we have to examine another problem whose solution will lead to an extension of ϑ_1 that will also be the basis for handling the dilemma.

5.2 First improvements: Penalizing frequently and recently occurring critical pairs

Suppose that an update run does not bring out any new members for N. This means that N does not change, and it is highly probable that the best ω which was used for this update run will still be the best in the next iteration. Therefore the GA may get stuck in a "local optimum", not on account of a deficiency of the GA itself, but on account of the inadequacy of the "testing environment" provided by N (and P) which is only a simulation of the "real thing", namely the first T (milli-) seconds of a proof as opposed to a complete proof. Moreover, such a "local optimum" can be substantially inferior to the global optimum. The essential problem consists in ϑ_1 considering only the *number of elements* in each $\mathcal{I}_j^\omega(N)$, not the elements *themselves*. It is especially unable to favor a $\overline{\omega}$ which is associated with different $\mathcal{I}_1^{\overline{\omega}}(N), \ldots, \mathcal{I}_m^{\overline{\omega}}(N)$ and has the same or may be an even worse fitness rating than ω. But such a $\overline{\omega}$ might actually shake the search up by admitting alternative negative critical pairs.

It is also worthwhile noting that a situation where there are no changes of N during an update run quite often originates from the generation of few, but "time intensive" negative critical pairs. (Time intensive critical pairs denote critical pairs that entail a lot of reductions and the generation of many further critical pairs when they are activated to become rules or equations.) Considering the fact that an update run—lasting only a limited time T—corresponds to the initial part of a proof attempt, we have to "predict" the subsequent behavior. From our own experience (and most readers will agree) such a situation as just described is in most cases disadvantageous since a heuristic that produces many unnecessary critical pairs, but does so fast, is in general less prone to wasting a lot of time than a heuristic that is busy with fewer, but more time intensive negative critical pairs.

Considering all this, it seems recommendable to introduce a penalty for negative critical pairs that keep on appearing in update runs in order to allow the GA to examine alternatives. This measure can be realized by associating with each element of N a number indicating how often the respective negative critical pair occurred during an update run. That is, we have a μ with $1 \leq \mu(\langle u, v\rangle) \leq i_c$ for all $\langle u, v\rangle \in N$, where i_c is the number of the current iteration and $\mu(\langle u, v\rangle)$ denotes the number of occurrences of $\langle u, v\rangle \in N$ during the past i_c update runs. (Recall that an update run is executed at the beginning of each iteration.) Then we use $\sum_{\langle u,v\rangle \in \mathcal{I}_j^{\omega}(N)} \mu(\langle u, v\rangle)$ instead of $|\mathcal{I}_j^{\omega}(N)|$. (Note that $|\mathcal{I}_j^{\omega}(N)| \leq \sum_{\langle u,v\rangle \in \mathcal{I}_j^{\omega}(N)} \mu(\langle u, v\rangle)$ and $\mathcal{I}_j^{\omega}(N) \subseteq N$.) Hence we have

$$\vartheta_2(\omega) = \sum_{j=1}^{m} \sum_{\langle u,v\rangle \in \mathcal{I}_j^{\omega}(N)} \mu(\langle u, v\rangle).$$

It must be outlined that any update run will cause a change of μ unless it produces no negative critical pairs at all. (But in that case the respective ω used for the update run should have found a proof if T is not too small.) Furthermore we achieved the goal to penalize frequently occurring negative critical pairs. As a consequence, a heuristic ω with possibly more, but less frequently occurring negative critical pairs in the respective sets $\mathcal{I}_1^{\omega}(N), \ldots, \mathcal{I}_m^{\omega}(N)$ will finally be given a better evaluation than a ω' that has less, but more frequently occurring negative critical pairs in its $\mathcal{I}_1^{\omega'}(N), \ldots, \mathcal{I}_m^{\omega'}(N)$. μ can hence be regarded as a meter for relevance or irrelevance of negative critical pairs, because a negative critical pair $\langle u, v\rangle$ which is generated over and over again although the ω used for the update runs changed must be considered more relevant than a negative critical pair $\langle u', v'\rangle$ possibly generated only once. So, $\mu(\langle u, v\rangle)$ and $\mu(\langle u', v'\rangle)$ will differ more and more as the number of iterations increases, and the in this sense more relevant $\langle u, v\rangle$ will have the desired property of influencing ϑ_2 with augmented intensity compared to $\langle u', v'\rangle$.

The use of μ as just described already enables us to handle the problematic related to relevance and irrelevance of negative critical pairs in a satisfactory way. But we can still refine ϑ_2. Considering the fact that the best ω—which is used for update runs—becomes more and more adapted from iteration to iteration, it is obvious that negative critical pairs encountered in recent iterations are more important than those generated in initial iterations. A function ψ takes this into account by having the value $\mu(\langle u,v\rangle)$ "fade away" as long as $\langle u,v\rangle \in N$ does not reappear during update runs, whereas $\mu(\langle u,v\rangle)$ remains unchanged if $\langle u,v\rangle$ occurred in the most recent update run. We define $\psi\colon Term(\mathcal{F},\mathcal{V})^2 \to \mathbb{N}$ as

$$\psi(\langle u,v\rangle) = \beta\left(\mu(\langle u,v\rangle) - (i_c - i_{mr}(\langle u,v\rangle))\right)$$

where

$$\beta : \mathbf{Z} \to \mathbb{N}, \quad \beta(x) = \begin{cases} x, & x > 0 \\ 1, & \text{otherwise} \end{cases}$$

As above, i_c is the number of the current iteration. $i_{mr}(\langle u,v\rangle)$ denotes the number of the iteration whose associated update run $\langle u,v\rangle$ last occurred in, i.e., $1 \le i_{mr}(\langle u,v\rangle) \le i_c$ for all $\langle u,v\rangle \in N$. Thus we have

$$\vartheta_3(\omega) = \sum_{j=1}^{m} \sum_{\langle u,v\rangle \in \mathcal{I}_j^\omega(N)} \psi(\langle u,v\rangle)$$

5.3 Further refinements

A further refinement of ϑ_3 can be accomplished by distinguishing elements of $\mathcal{I}_j^\omega(N)$ with a weight equal to $\omega(\langle u_j,v_j\rangle)$ and elements with a smaller weight. Assuming the existence of $\langle u,v\rangle$ (cp. what we said above), the latter always are obstacles during search, because they will definitely be preferred to $\langle u_j,v_j\rangle$, whereas the former do not *necessarily* cause a delay (recall that critical pairs with equal weights are processed according to the FIFO-strategy, see section 4). The following extensions $\varphi_1^\omega,\ldots,\varphi_m^\omega$ of μ implement this aspect.

$$\varphi_j^\omega(\langle u,v\rangle) = \begin{cases} \left\lfloor \frac{\mu(\langle u,v\rangle)+1}{2} \right\rfloor, & \text{if } \omega(\langle u,v\rangle) = \omega(\langle u_j,v_j\rangle) \\ \mu(\langle u,v\rangle), & \text{otherwise} \end{cases} \qquad 1 \le j \le m, \ \ \langle u,v\rangle \in \mathcal{I}_j^\omega(N)$$

Consequently, instead of one ψ we have

$$\psi_j'^{\omega}(\langle u,v\rangle) = \beta\left(\varphi_j^\omega(\langle u,v\rangle) - (i_c - i_{mr}(\langle u,v\rangle))\right), \quad 1 \le j \le m, \quad \langle u,v\rangle \in \mathcal{I}_j^\omega(N)$$

with i_c, i_{mr} and β as before, and

$$\vartheta_4(\omega) = \sum_{j=1}^{m} \sum_{\langle u,v\rangle \in \mathcal{I}_j^\omega(N)} \psi_j'^{\omega}(\langle u,v\rangle)$$

So far we have not taken into account the position of the members of N or P in the derivation graph. Since this property is rather independent of the particular ω used to update N, it appears profitable to make use of it as an additional means to support our efforts to distinguish relevant and irrelevant (negative) critical pairs. We chose the *depth of a critical pair* as the quantity describing its position in the derivation graph in the most expressive and concise way.

Definition 5.2 (depth) *Let $\langle u,v\rangle$ be a critical pair, $l_1 \sim_1 r_1$, $l_2 \sim_2 r_2$ rules or equations, i.e., $\sim_1, \sim_2 \in \{\rightarrow, =\}$, and $\mathcal{A} = (Ax, Th)$ the current problem. The depth $\delta : Term(\mathcal{F}, \mathcal{V})^2 \rightarrow I\!N$ is defined as*

$$\delta(\langle u,v\rangle) = \begin{cases} 0, & \text{if } u = v \in Ax \\ \delta(l_1 \sim_1 r_1) + 1, & \langle u,v\rangle \text{ obtained by reducing } l_1 \sim_1 r_1 \\ \max(\{\delta(l_1 \sim_1 r_1), \delta(l_2 \sim_2 r_2)\}) + 1, & \langle u,v\rangle \text{ crit. pair of } l_1 \sim_1 r_1 \text{ and } l_2 \sim_2 r_2 \end{cases}$$

The depth of a rule or equation equals the depth of the critical pair the rule or equation stems from.

For every $\langle u_j, v_j\rangle \in P$, $\delta(\langle u_j, v_j\rangle)$ is determined a priori using the known proof $\mathcal{P}$ and stays fixed during the whole adaptation procedure. For a $\langle u,v\rangle \in N$, $\delta(\langle u,v\rangle)$ is—if necessary—updated resp. initialized after each update run.

Let us now examine what kind of information we can possibly obtain considering the depth of a $\langle u,v\rangle \in \mathcal{I}_j^\omega(N)$ compared to the depth of $\langle u_j, v_j\rangle \in P$. Recall that we intend to employ δ as an additional means for estimating relevance or irrelevance. Let $d = \delta(\langle u_j, v_j\rangle)$ for some $1 \leq j \leq m$. It is obvious that, on the one hand, the probability that $\langle u,v\rangle \in \mathcal{I}_j^\omega(N)$ does not exist before $\langle u_j, v_j\rangle$ is chosen (and hence is *irrelevant*) increases the more $\delta(\langle u,v\rangle)$ exceeds d. On the other hand, the probability that $\langle u,v\rangle \in \mathcal{I}_j^\omega(N)$ does exist before $\langle u_j, v_j\rangle$ is chosen (and hence is *irrelevant*) increases as $\delta(\langle u,v\rangle)$ decreases.

We decided to incorporate these considerations by making use of the difference $\delta(\langle u_j, v_j\rangle) - \delta(\langle u,v\rangle)$. Hence, we obtain $\psi_1^\omega, \ldots, \psi_m^\omega$ in place of $\psi_1'^\omega, \ldots, \psi_m'^\omega$:

$$\psi_j^\omega(\langle u,v\rangle) = \beta\Big(\underbrace{(\varphi_j^\omega(\langle u,v\rangle) - (i_c - i_{mr}(\langle u,v\rangle)))}_{\text{occurrence component}} \cdot \underbrace{\beta(\delta(\langle u_j, v_j\rangle) - \delta(\langle u,v\rangle) + \mathcal{D})}_{\text{depth component}}\Big)$$

The value of $\psi_j^\omega(\langle u,v\rangle)$, $1 \leq j \leq m$, becomes the bigger the more $\delta(\langle u_j, v_j\rangle)$ exceeds $\delta(\langle u,v\rangle)$. $\mathcal{D} \in \mathbf{Z}$ together with β control the effect of the difference $\delta(\langle u_j, v_j\rangle) - \delta(\langle u,v\rangle)$.

5.4 The final fitness function

The last refinement attempts to estimate the (detrimental) influence of *relevant* negative critical pairs which did not (yet) occur during update runs. Once again it must be emphasized that relevance is not a global property, but must be seen w.r.t. a particular ω and a $\langle u_j, v_j\rangle \in P$. Recall that, due to the fact that update runs are performed using only one (the best) ω' of the current population, it is almost certain that, given any ω and a $\langle u_j, v_j\rangle \in P$, there will be relevant negative critical pairs w.r.t. ω and $\langle u_j, v_j\rangle \in P$ which are not (yet) in N.

A relevant negative critical pair $\langle u,v\rangle$ w.r.t. a given ω and a $\langle u_j, v_j\rangle \in P$ satisfies $\omega(\langle u,v\rangle) \leq \omega(\langle u_j, v_j\rangle)$ (cp. definition 5.1). So, the chance to encounter such a relevant negative critical pairs grows with $\omega(\langle u_j, v_j\rangle)$. In general, the number of relevant negative critical pairs $\langle u,v\rangle$ w.r.t. ω and $\langle u_j, v_j\rangle \in P$ does not depend linearly on $\omega(\langle u_j, v_j\rangle)$.

As a matter of fact, this number may even increase exponentially. As a consequence, the probability that a negative critical pair not (yet) in N occurs during a proof attempt using ω (and hence is relevant w.r.t. ω and $\langle u_j, v_j \rangle$) grows over-proportionally with $\omega(\langle u_j, v_j \rangle)$, $1 \leq j \leq m$. This should be reflected by the fitness function. One problem here is the fact that ω is not "normalized", i.e., depending on the parameters one ω may generally produce lower or higher weights than some other ω' without necessarily being inherently different. Therefore a quantity κ_j^ω based on $|\mathcal{I}_j^\omega(N)|$, $1 \leq j \leq m$, is introduced for this purpose. This choice is motivated by $|\mathcal{I}_j^\omega(N)|$ being "more normalized" than $\omega(\langle u_j, v_j \rangle)$, but still providing a reasonable means for estimating the probability that additional negative critical pairs occur: The connection between $|\mathcal{I}_j^\omega(N)|$ and $\omega(\langle u_j, v_j \rangle)$ is revealed by the observation that it is the more likely to run into a negative critical pair $\langle u, v \rangle \notin N$ with $\omega(\langle u, v \rangle) \leq \omega(\langle u_j, v_j \rangle)$ the more elements already are in $\mathcal{I}_j^\omega(N)$, and as before, this likelihood grows over-proportionally with $|\mathcal{I}_j^\omega(N)|$. κ_j^ω—based on $|\mathcal{I}_j^\omega(N)|$—precludes those members of $\mathcal{I}_j^\omega(N)$ considered irrelevant according to the criteria elaborated for ϑ_3 and ϑ_4 above.

$$\kappa_j^\omega = \sum_{\langle u,v \rangle \in \mathcal{I}_j^\omega(N)} \pi\left(\varphi_j^\omega(\langle u, v \rangle) - (i_c - i_{mr}(\langle u, v \rangle))\right)$$

where

$$\pi : \mathbf{Z} \to \{0,1\}, \quad \pi(x) = \begin{cases} 1, & x > 0 \\ 0, & \text{otherwise.} \end{cases}$$

(*Note*: $\kappa_j^\omega \leq |\mathcal{I}_j^\omega(N)|$.)

Let furthermore $\zeta_j^\omega = \sum_{\langle u,v \rangle \in \mathcal{I}_j^\omega(N)} \psi_j^\omega(\langle u, v \rangle)$. ζ_j^ω is a measure for the "difficulties" we (probably) have to face when using ω and trying to reach $\langle u_j, v_j \rangle \in P$. If ζ_j^ω grows these difficulties (are thought to) augment. The ideas presented above are realized by

$$\vartheta_A(\omega) = \sum_{j=1}^{m} \alpha\left(\kappa_j^\omega, \zeta_j^\omega\right)$$

where

$$\alpha(x,y) = \left\lfloor \frac{x^2 \cdot y}{\mathcal{C}} \right\rfloor, \quad \mathcal{C} \in \mathbb{N}.$$

α is of course not only inspired by the above considerations, but also by experimenting.

The final version of the fitness function ϑ is a lexicographic combination of ϑ_A, ϑ_B (corresponds to ϑ_4), ϑ_C (average weight of all positive critical pairs) and ϑ_D (the negative average weight of all currently known negative critical pairs):

$$\begin{aligned} \vartheta_B(\omega) &= \sum_{j=1}^{m} \zeta_j^\omega \\ \vartheta_C(\omega) &= \frac{\sum_{j=1}^{m} \omega(\langle u_j, v_j \rangle)}{m} \qquad (m > 0) \\ \vartheta_D(\omega) &= -\frac{\sum_{\langle u,v \rangle \in N} \omega(\langle u, v \rangle)}{|N|} \qquad (N \neq \emptyset) \end{aligned}$$

Hence $\vartheta(\omega) = (\vartheta_A(\omega), \vartheta_B(\omega), \vartheta_C(\omega), \vartheta_D(\omega))$, and $\vartheta(\omega) <_{lex} \vartheta(\omega')$ signifies that ω is considered to be better than ω'. $<_{lex}$ is the lexicographic comparison from left to right using the usual ordering $<$ on natural numbers.

Having developed the fitness function ϑ we can conclude this section. It remains to remark that ϑ is a *dynamically changing fitness function* because of the periodic updates of the set N which ϑ is in part based on.

The next section documents our experimental results.

6 Experimental results

In this section we document the results achieved with our method for learning proof heuristics. Recall that the aim of learning is to find parameters for a heuristic $\mathcal{W}$ (*add*, *max*, *gt*, *occnest* or *linpol*, cp. section 4) so that a proof problem $\mathcal{A}'$ can be solved with as little redundancy as possible when employing the learned instance ω of $\mathcal{W}$. (Criteria for the minimality of redundancy are, for instance, the number of rules, equations and critical pairs generated, the number of reductions and—last but not least—the time spent on finding a proof.) The learning process is guided by a known proof $\mathcal{P}$ of $\mathcal{A}'$, which also makes our approach a method for (indirect) proof reuse. Of course, the major goal of any kind of proof reuse does not consist in merely speeding up the search for a known proof. The broader expectations are that a problem $\mathcal{A}$ similar to $\mathcal{A}'$ can be solved (much) faster using an ω adapted to $\mathcal{A}'$ than with a default heuristic. For each set of (similar) problems we shall therefore also list the results produced by the default versions of the heuristics. As we shall see, we could not only establish salient speed-ups, but also succeeded in proving (completing) problems that were out of reach for DISCOUNT when using default heuristics.

In subsection 3.2 we sketched the working method of the adaptive procedure which accomplishes the learning of parameters of a given $\mathcal{W}$. The adaptive procedure performs iterations (during which the genetic algorithm performs n_c cycles). Each of these iterations starts with an update run that lasts at most T seconds. Update runs produce (new) negative facts (critical pairs) which influence the fitness function. Consequently, T is an important parameter of the adaptive procedure. So is $\mathcal{W}$, because the suitability of $\mathcal{W}$ to learn a given $\mathcal{P}$ depends on $\mathcal{W}$ itself. Some $\mathcal{W}$ may be "ideal" to learn a proof $\mathcal{P}$, while another $\mathcal{W}'$ is not, because the properties of $\mathcal{W}'$ cannot be made to agree sufficiently with the features of $\mathcal{P}$, i.e., redundancies cannot be reduced significantly. In the experimental stage we are currently in it is satisfactory to leave a judicious choice of T and $\mathcal{W}$ to the experimenter. (An obvious choice for T is a certain percentage of the time needed to find the proof $\mathcal{P}$. A good choice for $\mathcal{W}$ is the heuristic employed to find $\mathcal{P}$ or one that proved to be successful for a number of problems with the same axiomatization.) We shall investigate this problem more closely when its (complete) automation becomes necessary. For the moment, we content ourselves with the fact that adaptation can be successful provided that T and $\mathcal{W}$ have been chosen appropriately.

Our experiments were conducted as follows: Given a set of (intuitively) more or less similar problems $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and their associated proofs $\mathcal{P}_1, \ldots, \mathcal{P}_n$, we had the adaptive procedure perform ten iterations using an $\mathcal{A}_i$ and its associated proof $\mathcal{P}_i$ taken from (a subset of) $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ in order to adapt an also given $\mathcal{W}$ to $\mathcal{P}_i$. The 'time out' parameter T was also provided. Any instance ω of $\mathcal{W}$ employed during one of the ten update runs that allowed for a proof of $\mathcal{A}_i$ (within T seconds) was among the output of the adaptive procedure (cp. subsection 3.2). Subsections 6.1–6.4—each dealing with a different set of "similar" problems—only display the best of the ω obtained during the ten iterations that exhibit clearly distinct properties. (A complete list can be found in appendix B.) Every ω was not only tested w.r.t. the problem $\mathcal{A}_i$ it was adapted to, but also w.r.t. all "similar" problems $\mathcal{A}_1, \ldots, \mathcal{A}_n$, in order to find out if our broader expectations, namely the successful applicability of a learned ω to similar problems, are also fulfilled. (*Note*: In the following subsections, the results (instances) attained when adapting a heuristic $\mathcal{W}$ are only reported if they are not generally inferior to those obtained when adapting a different $\mathcal{W}'$.)

The entries in the bodies of the subsequent tables display run-times (in seconds) obtained on a SPARCstation 1. We shall explain by way of example how to interpret the tables. The run-times of the adaptive procedure to produce the instances of the heuristics reported in subsections 6.1–6.4 ranged in general between one and three minutes. Computation time was significantly higher when we chose $\mathcal{W} \equiv$ *linpol*, because *linpol* requires multiple precision integer arithmetic which is rather time consuming. For this reason we fell back on *linpol* only if the adaptation of any other heuristic did not produce satisfactory results. But we think that the time spent by the adaptive procedure is not an essential issue anyway, because our concept allows to carry through adaptation during the prover's "spare time" since it is completely independent of future target problems. Apart from that, some of the improvements do justify this effort (see in particular subsection 6.4) even if the time spent by the adaptive procedure is taken into account rigorously.

6.1 The first set of examples: non-associative rings

The first set of examples is taken from the theory of non-associative rings (see, for instance, [Sch66]). Three problems $\mathcal{A}_1^{NA}$, $\mathcal{A}_2^{NA}$ and $\mathcal{A}_3^{NA}$ were examined, where $\mathcal{A}_i^{NA} = (Ax^{NA}, Th_i^{NA})$ for $1 \leq i \leq 3$. The set of axioms Ax^{NA} is:

$$\begin{array}{rclrcl}
f(x,0) & = & x & h(x,h(y,y)) & = & h(h(x,y),y) \\
f(x,g(x)) & = & 0 & h(h(x,x),y) & = & h(x,h(x,y)) \\
f(f(x,y),z) & = & f(x,f(y,z)) & h(x,f(y,z)) & = & f(h(x,y),h(x,z)) \\
f(x,y) & = & f(y,x) & h(f(x,y),z) & = & f(h(x,z),h(y,z))
\end{array}$$

$$a(x,y,z) = f(h(h(x,y),z),g(h(x,h(y,z))))$$

and the three theorems represent the linearity of the associator a in all of its three arguments:

$$\begin{array}{lcl} Th_1^{NA} & \equiv & a(f(x,y),u,v) = f(a(x,u,v),a(y,u,v)) \\ Th_2^{NA} & \equiv & a(u,f(x,y),v) = f(a(u,x,v),a(u,y,v)) \\ Th_3^{NA} & \equiv & a(u,v,f(x,y)) = f(a(u,v,x),a(u,v,y)) \end{array}$$

The first four rows of table 6.1.1 show the results obtained using the default versions of our heuristics. The last three rows summarize the best results obtained by the adaptive procedure. $\omega_j^{(i)}$ was produced by the adaptive procedure when learning the proof of $\mathcal{A}_i^{NA}$. It corresponds to ω_j in table B.1.i (see appendix B) where also the remaining (inferior) ω's are listed. In every case we chose $\mathcal{W} \equiv gt$ and $T = 5sec$. The reduction ordering always was a LPO with precedence $a \succ_p h \succ_p g \succ_p f \succ_p 0$. An entry '$\infty$' indicates that no proof could be found within 3 hours.

	$\mathcal{A}_1^{NA}$	$\mathcal{A}_2^{NA}$	$\mathcal{A}_3^{NA}$
add	148.2s	148.3s	149.1s
max	∞	∞	∞
gt	7.59s	7.55s	7.71s
$occnest$	∞	∞	∞
$\omega_3^{(1)}$	0.337s	0.342s	0.337s
$\omega_2^{(2)}$	0.341s	0.349s	0.342s
$\omega_3^{(3)}$	0.336s	0.345s	0.337s

Table 6.1.1

Table 6.1.1 (and all the subsequent tables) can be read in the usual manner, i.e., look up the entry in the row labeled $\mathcal{H}$ and the column labeled $\mathcal{A}_i^{NA}$ in order to find out how long it took DISCOUNT to prove $\mathcal{A}_i^{NA}$ when using the default or adapted heuristic $\mathcal{H}$.

The results listed in tables 6.1 show that, no matter which theorem's proof was learned during adaptation, the respectively best parameter configurations yield speed-ups of more than 20, and—even more important to note—these improvements remain effective when employing them for proving the other theorems. So, our method easily captured the obvious similarity of Th_1^{NA}, Th_2^{NA} and Th_3^{NA} without having to pin down the nature of this similarity. It is also worth noting that (in all three cases) the best parameter configuration allows for a proof with a negligible amount of redundancy.

6.2 The second set of examples: propositional logic

The second set of examples stems from [Ta56]. The set of axioms Ax^{PL} is given by

$$\begin{array}{rcl} c(t,x) & = & x \\ c(x,c(y,x)) & = & t \\ c(n(n(x)),x) & = & t \\ c(x,n(n(x))) & = & t \end{array} \qquad \begin{array}{rcl} & & \\ c(c(x,c(y,z)),c(c(x,y),c(x,z))) & = & t \\ c(c(x,c(y,z)),c(y,c(x,z))) & = & t \\ c(c(x,y),c(n(y),n(x))) & = & t \end{array}$$

and is an equational axiomatization of the propositional logic. ('c' corresponds to 'implication', 'n' to 'not' and 't' to 'true'.) Consequently, the theorems are tautologies of the propositional logic.

$$\begin{array}{llll}
Th_1^{PL} \equiv c(x,c(n(x),y)) = t & & Th_6^{PL} \equiv c(n(x),c(x,n(y))) = t \\
Th_2^{PL} \equiv c(n(c(n(x),y)),c(y,n(z))) = t & & Th_7^{PL} \equiv c(n(c(x,n(y))),y) = t \\
Th_3^{PL} \equiv c(n(t),x) = t & & Th_8^{PL} \equiv c(n(c(n(x),n(y))),y) = t \\
Th_4^{PL} \equiv c(n(x),c(x,y)) = t & & Th_9^{PL} \equiv c(x,c(n(c(y,n(y))),y)) = t \\
Th_5^{PL} \equiv c(x,c(n(x),n(y))) = t & & Th_{10}^{PL} \equiv c(x,c(n(y),c(y,n(z)))) = t
\end{array}$$

Table 6.2.1 shows the results of the default heuristics. For these problems the default versions of *add*, *max* and *gt* agree completely. Therefore only the results produced by *add* and *occnest* are listed. We always used a LPO with precedence $c \succ_p n \succ_p t$.

	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
add	278.2s	16.1s	6.1s	276.1s	13.6s	13.8s	6.1s	6.1s	6.1s	13.8s
occnest	13.8s	30.2s	6.3s	13.9s	14.5s	14.6s	7.4s	9.7s	8.8s	56.1s

Table 6.2.1: *default heuristics*

Table 6.2.2 lists the most prominent results achieved when adapting $\mathcal{W} \equiv$ *occnest*. ($\omega_i^{(j)}$ is an instance of $\mathcal{W}$ generated when learning a proof of $\mathcal{A}_j^{PL} = (Ax^{PL}, Th_j^{PL})$. It corresponds to ω_i in table B.2.2.j of appendix B which shows all instances of $\mathcal{W}$ produced in the course of ten iterations of the adaptive procedure.) We set $T = 5sec.$, except when learning the proofs of $\mathcal{A}_1^{PL}$, $\mathcal{A}_4^{PL}$ and $\mathcal{A}_8^{PL}$, where we raised T to $9sec.$ in order to find instances of $\mathcal{W}$ that enabled DISCOUNT to succeed before time out. Note that increasing T not only allows for detecting instances ω that require more time to find a proof but also changes the search behavior, because more negative critical pairs occur during prolonged update runs (which affect the fitness function). At the one hand, increasing T may be a benefit, because update runs can then provide more accurate data *with respect to an ω employed during an update run.* On the other hand, this may aggravate the problematic connected with irrelevant (negative) critical pairs for all those ω (constituting the majority) which are not used for update runs. (The choice of T—as we already stated in the introduction to this section—is so far left to the (human) experimenter.) Tables 6.2.2 and 6.2.3 can again be interpreted in the usual manner. (E.g., in table 6.2.2, $\omega_5^{(6)}$ enabled DISCOUNT to prove $\mathcal{A}_2^{PL}$ in $2.2sec.$) An entry '—' indicates that no proof could be found in a time comparable to the run-times achieved when using a default heuristic or another adapted heuristic.

Although there is (intuitively) little resemblance between the above theorems (except for Th_1^{PL} and Th_4^{PL} and instances Th_5^{PL} and Th_6^{PL} thereof) tables 6.2.2 and 6.2.3 reveal that certain groups of theorems can be proved in a similar fashion. We shall now take a closer look at parts of the results presented in these two tables.

First of all we have to clarify an observation in connection with Th_1^{PL} and Th_4^{PL} and their respective instances Th_5^{PL} and Th_6^{PL}. From table 6.2.2 we can see that it takes DISCOUNT about twice as long to prove the instances $\mathcal{A}_5^{PL}$ and $\mathcal{A}_6^{PL}$ when

	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
$\omega_1^{(1)}$	7.9s	26.2s	8.4s	7.8s	16.7s	16.4s	9.2s	14.6s	11.6s	18.2s
$\omega_8^{(2)}$	—	2.2s	—	—	1.8s	1.9s	—	—	15.3s	3.5s
$\omega_2^{(3)}$	—	—	4.4s	—	—	—	9.5s	15.4s	10.8s	—
$\omega_2^{(4)}$	7.8s	26.3s	7.8s	7.8s	16.6s	16.5s	9.2s	14.7s	11.1s	18.2s
$\omega_2^{(5)}$	—	2.2s	—	—	1.8s	1.9s	—	—	15.3s	3.5s
$\omega_4^{(5)}$	—	11.8s	—	—	2.1s	2.2s	—	—	2.4s	3.5s
$\omega_1^{(6)}$	—	12.7s	—	—	2.1s	2.2s	—	—	2.1s	3.5s
$\omega_5^{(6)}$	—	2.2s	—	—	1.8s	1.9s	—	—	12.8s	2.5s
$\omega_1^{(7)}$	—	—	18.6s	—	28.2s	28.0s	4.1s	5.7s	2.7s	24.5s
$\omega_2^{(7)}$	—	19.9s	—	—	30.0s	29.8s	2.7s	3.8s	2.4s	—
$\omega_1^{(8)}$	—	17.7s	—	—	27.7s	28.0s	7.4s	7.7s	1.7s	21.1s
$\omega_4^{(8)}$	—	—	7.7s	—	26.0s	25.8s	5.5s	6.9s	6.0s	23.4s
$\omega_1^{(9)}$	—	19.6s	—	—	30.1s	29.9s	3.0s	4.2s	2.3s	—
$\omega_2^{(9)}$	—	11.9s	—	—	27.7s	28.0s	17.9s	19.2s	1.7s	21.1s
$\omega_1^{(10)}$	—	12.3s	—	—	3.2s	3.3s	17.9s	18.7s	2.5s	2.8s
$\omega_2^{(10)}$	—	2.4s	—	—	2.0s	2.1s	—	—	13.3s	2.6s

Table 6.2.2: $\mathcal{W} \equiv occnest$

	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
$\omega_1^{(1)}$	4.9s	5.4s	1.3s	4.8s	3.1s	3.2s	—	—	21.3s	3.2s
$\omega_1^{(2)}$	29.1s	4.3s	33.4s	28.9s	0.8s	0.9s	—	—	—	0.9s
$\omega_1^{(3)}$	24.2s	27.1s	1.7s	24.0s	15.4s	15.7s	2.0s	1.7s	1.8s	18.8s
$\omega_2^{(3)}$	—	—	0.7s	—	—	—	0.7s	0.7s	0.7s	—
$\omega_1^{(4)}$	4.9s	5.4s	1.3s	4.8s	3.2s	3.3s	—	—	—	3.2s
$\omega_2^{(5)}$	20.0s	29.9s	24.2s	20.5s	1.0s	1.6s	—	—	—	1.0s
$\omega_5^{(5)}$	—	—	—	—	0.5s	0.5s	—	—	—	0.5s
$\omega_7^{(6)}$	6.9s	6.2s	9.1s	6.9s	1.9s	1.9s	—	—	—	2.0s
$\omega_1^{(7)}$	—	24.2s	1.6s	—	9.3s	9.5s	1.6s	1.6s	1.6s	9.5s
$\omega_4^{(8)}$	9.8s	11.0s	0.8s	9.6s	5.2s	5.3s	0.9s	0.8s	0.8s	5.4s
$\omega_2^{(9)}$	—	21.0s	2.0s	—	14.3s	14.5s	2.4s	2.4s	2.4s	14.5s
$\omega_7^{(9)}$	—	—	0.6s	—	16.5s	16.6s	0.6s	0.6s	0.7s	16.6s
$\omega_2^{(10)}$	5.7s	7.5s	5.0s	5.6s	2.3s	2.4s	—	—	—	2.4s
$\omega_4^{(10)}$	—	—	—	—	0.7s	0.7s	—	—	—	0.7s

Table 6.2.3: $\mathcal{W} \equiv$ *linpol*

using $\omega_1^{(1)}$ or $\omega_2^{(4)}$, which learned a proof of $\mathcal{A}_1^{PL}$ and $\mathcal{A}_4^{PL}$, respectively. This at first sight confusing and contradictory finding is explained by the goal oriented character of *occnest* which causes changes in the search behavior depending on the current goal. (This is not the case when adapting $\mathcal{W} \equiv$ *linpol* as we can see from table 6.2.3.)

Table 6.2.2 also displays the all-pervasive dilemma between *specialization* (i.e., being tailored to the needs of a very narrow group of problems) and *generality* (i.e., being more generally applicable with (moderate) success). Consider again the rows headed by $\omega_1^{(1)}$ and $\omega_2^{(4)}$. When applying these heuristics DISCOUNT can prove all ten theorems with moderate improvements and moderate deterioration. Hence $\omega_1^{(1)}$ and $\omega_2^{(4)}$ can be viewed as "general", whereas $\omega_2^{(5)}$ and $\omega_4^{(5)}$, for instance, are more "specialized", because they cause substantial improvements for half the problems which come at the expense of (substantial) deterioration of the remaining problems.

$\omega_2^{(5)}$ and $\omega_4^{(5)}$ reveal another phenomenon which consists in a kind of "incompatibility" between $\mathcal{A}_2^{PL}$ and $\mathcal{A}_9^{PL}$ as far as proving them using a heuristic that learned a proof of $\mathcal{A}_5^{PL}$ is concerned: Reducing the run-time for proving either $\mathcal{A}_2^{PL}$ or $\mathcal{A}_9^{PL}$ always entails increasing the run-time for proving the other problem (see also table B.2.2.5). The same goes for $\omega_1^{(6)}$ and $\omega_5^{(6)}$.

Basically, the same effects can also be observed in table 6.2.3, where we list the results obtained when adapting $\mathcal{W} \equiv$ *linpol*. (As before, the complete results are compiled in tables B.2.3.1–B.2.3.10.) The benefit and drawback of specialization becomes particularly apparent in the row of table 6.2.3 headed by $\omega_5^{(5)}$. The adapted heuristic $\omega_5^{(5)}$ allows for extremely fast proofs of $\mathcal{A}_5^{PL}$, $\mathcal{A}_6^{PL}$ and $\mathcal{A}_{10}^{PL}$, but is unsuitable for the remaining problems.

It is important to note that $\omega_4^{(8)}$ (i.e., an instance of *linpol* that learned a proof of $\mathcal{A}_8^{PL}$) allows for proofs of all the problems with moderate to significant improvements *without* a change for the worse.

Although both *occnest* and *linpol* differ in their learning capability, we still can recognize in both tables that some of the problems must be very similar, because the heuristics adapted to them behave similarly. E.g., in both table 6.2.2 and table 6.2.3 the results obtained when learning $\mathcal{A}_1^{PL}$ or $\mathcal{A}_4^{PL}$ are almost identical (cf. $\omega_1^{(1)}$ and $\omega_2^{(4)}$ in table 6.2.2, $\omega_1^{(1)}$ and $\omega_1^{(4)}$ in table 6.2.3).

The above tables show that the danger of causing a change for the worse by employing adapted heuristics (as compared to default heuristics) is omnipresent. Therefore means for assessing (or better "guessing") the benefit of a learned heuristic for handling a new problem *a priori* are indispensable. Evaluation of the data collected in these tables allows an *a posteriori* estimation of profitableness, but may also serve as a statistic foundation of a priori estimations.

6.3 The third set of examples: Lattice ordered groups

The equational axiomatization of lattice ordered groups Ax^{LOG} (cp. also [Fu94]) is as follows. (Note: u and l represent the *least upper* resp. *greatest lower bound.*) For details on (lattice) ordered groups see, for instance, [KK74].

$$\begin{array}{rclrcl}
l(x,y) & = & l(y,x) & & & \\
u(x,y) & = & u(y,x) & f(f(x,y),z) & = & f(x,f(y,z)) \\
l(l(x,y),z) & = & l(x,l(y,z)) & f(1,x) & = & x \\
u(u(x,y),z) & = & u(x,u(y,z)) & f(i(x),x) & = & 1 \\
u(x,x) & = & x & f(x,u(y,z)) & = & u(f(x,y),f(x,z)) \\
l(x,x) & = & x & f(x,l(y,z)) & = & l(f(x,y),f(x,z)) \\
u(x,l(x,y)) & = & x & f(u(x,y),z) & = & u(f(x,z),f(y,z)) \\
l(x,u(x,y)) & = & x & f(l(x,y),z) & = & l(f(x,z),f(y,z))
\end{array}$$

The theorems are:

$$\begin{array}{lcl}
Th_{1a}^{LOG} & \equiv & u(i(x),i(y)) = i(x) \rightarrow u(x,y) = y \\
Th_{1b}^{LOG} & \equiv & l(i(x),i(y)) = i(x) \rightarrow l(x,y) = y \\
Th_{2a}^{LOG} & \equiv & u(x,y) = x \rightarrow u(i(x),i(y)) = i(y) \\
Th_{2b}^{LOG} & \equiv & l(x,y) = x \rightarrow l(i(x),i(y)) = i(y) \\
Th_{3a}^{LOG} & \equiv & u(1,x) = x, u(1,y) = y, u(1,z) = z, 1 = l(x,y) \rightarrow l(x,f(y,z)) = l(x,z) \\
Th_{3b}^{LOG} & \equiv & l(1,x) = 1, l(1,y) = 1, l(1,z) = 1, 1 = l(x,y) \rightarrow l(x,f(y,z)) = l(x,z)
\end{array}$$

heuristic	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
add	∞	∞	44.8s	44.9s
max	∞	∞	870s	869s
gt	∞	∞	358s	357s
occnest	∞	∞	5.2s	3.8s

Table 6.3.1: *default versions*

	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
$\omega_7^{(1a)}$	0.294s	0.295s	0.370s	0.399s
$\omega_1^{(1b)}$	0.253s	0.236s	0.251s	0.222s
$\omega_8^{(2a)}$	0.303s	0.300s	0.324s	0.498s
$\omega_2^{(2b)}$	0.538s	0.637s	0.638s	0.855s

Table 6.3.2: $\mathcal{W} \equiv$ *add*

	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
$\omega_6^{(1a)}$	0.166s	0.174s	0.226s	0.209s
$\omega_1^{(1b)}$	0.187s	0.189s	0.208s	0.210s
$\omega_1^{(2a)}$	0.198s	0.197s	0.226s	0.211s
$\omega_1^{(2b)}$	0.204s	0.203s	0.218s	0.215s

Table 6.3.3: $\mathcal{W} \equiv$ *max*

	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
$\omega_3^{(1a)}$	0.381s	0.367s	0.215s	0.229s
$\omega_3^{(1b)}$	0.295s	0.274s	0.447s	0.553s
$\omega_7^{(2a)}$	1.174s	1.248s	0.990s	1.042s
$\omega_3^{(2b)}$	1.909s	1.953s	1.188s	1.187s

Table 6.3.4: $\mathcal{W} \equiv gt$

	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
$\omega_1^{(1a)}$	0.214s	0.292s	0.275s	0.226s
$\omega_1^{(1b)}$	0.210s	0.231s	0.318s	0.561s
$\omega_2^{(2a)}$	0.207s	0.220s	0.333s	0.324s
$\omega_4^{(2b)}$	1.758s	0.421s	0.733s	0.439s

Table 6.3.5: $\mathcal{W} \equiv$ *occnest*

Let $\mathcal{A}_{\mathcal{L}}^{LOG} = (Ax^{LOG}, Th_{\mathcal{L}}^{LOG})$ for $\mathcal{L} \in \{1a, 1b, 2a, 2b, 3a, 3b\}$. It is quite apparent that the problems in $\mathcal{M}_1 = \{\mathcal{A}_{1a}^{LOG}, \mathcal{A}_{1b}^{LOG}, \mathcal{A}_{2a}^{LOG}, \mathcal{A}_{2b}^{LOG}\}$ are similar. The same goes for the problems in $\mathcal{M}_2 = \{\mathcal{A}_{3a}^{LOG}, \mathcal{A}_{3b}^{LOG}\}$. A problem in $\mathcal{M}_1$, however, does not bear any resemblance to a problem in $\mathcal{M}_2$. Therefore we shall first consider the problems in $\mathcal{M}_1$ and then separately examine those in $\mathcal{M}_2$.

Table 6.3.1 lists the run-times of DISCOUNT w.r.t. the problems in $\mathcal{M}_1$ when using the default heuristics. An entry '∞' indicates that no proof was found within 3 hours, or DISCOUNT had to abort due to memory shortage. The results obtained when using adapted heuristics (also w.r.t. the problems in $\mathcal{M}_1$) can be found in tables 6.3.2–6.3.5. These tables display the best instances of *add*, *max*, *gt* and *occnest*, respectively, found by the adaptive procedure. An $\omega_j^{(\mathcal{L})}$ in table 6.3.i was produced when adapting the heuristic $\mathcal{W}$ designated at the bottom of that table to $\mathcal{A}_{\mathcal{L}}^{LOG}$. $\omega_j^{(\mathcal{L})}$ corresponds to ω_j in table B.3.i.$\mathcal{L}$ (see appendix B). Table B.3.i.$\mathcal{L}$ also shows all the other instances of the respective $\mathcal{W}$ generated in the course of the ten iterations performed by the adaptive procedure. In all cases we set $T = 5$ seconds, and the reduction ordering was a LPO with precedence $i \succ_p f \succ_p l \succ_p u \succ_p 1$. (*Note*: If $\mathcal{W}$ had to learn a proof of $\mathcal{A}_{1a}^{LOG}$ or $\mathcal{A}_{1b}^{LOG}$, the respective proof—which could not be found using our default heuristics (cp. table 6.3.1)—was nevertheless provided by DISCOUNT by applying the *teamwork method* ([AD93], [De93]). See also [DF94].) The most remarkable accomplishments of adaptation can be viewed in the third and fourth rows of tables 6.3.2–6.3.5. These rows show the results of the best instances of the respective $\mathcal{W}$ generated when learning a proof of $\mathcal{A}_{2a}^{LOG}$ (third row) or $\mathcal{A}_{2b}^{LOG}$ (fourth row). These proofs can—as table 6.3.1 reveals—be found by any of the default heuristics (more or less quickly). But with these learned instances DISCOUNT can also prove $\mathcal{A}_{1a}^{LOG}$ and $\mathcal{A}_{1b}^{LOG}$. Thus, the horizon of DISCOUNT is clearly broadened due to solving previously "unsolvable" problems, and this has been made possible by learning. The first and second rows exhibit salient speed-ups. Furthermore, it is important to note that the properties of the best instances do not vary significantly, no matter which heuristic $\mathcal{W}$ was adapted nor which problem it was adapted to.

Note that—on account of the duality of u and l in the axioms as well as to some extent in the theorems—we reversed the roles of u and l when applying $\omega_j^{(1a)}$ or $\omega_j^{(2a)}$ ($\omega_j^{(1b)}$ or $\omega_j^{(2b)}$) to $\mathcal{A}_{1b}^{LOG}$ or $\mathcal{A}_{2b}^{LOG}$ ($\mathcal{A}_{1a}^{LOG}$ or $\mathcal{A}_{2a}^{LOG}$). 'Reversing the roles' here simply means that the values of the parameters w_u and w_l are exchanged. This obvious measure is indispensable for success in these cases.

To conclude this subsection we briefly examine the problems in $\mathcal{M}_2$. The first four rows of table 6.3.6 below display the performance of DISCOUNT using default heuristics. Rows five and six show the results obtained when employing the best instance of $\mathcal{W} \equiv$ *occnest*[6] adapted to $\mathcal{A}_{3a}^{LOG}$ and $\mathcal{A}_{3b}^{LOG}$, respectively. ($\omega_j^{(\mathcal{L})}$ corresponds to ω_j in table B.3.6.$\mathcal{L}$.)

	$\mathcal{A}_{3a}^{LOG}$	$\mathcal{A}_{3b}^{LOG}$
add	209.1s	207.1s
max	∞	∞
gt	6157s	∞
occnest	19.6s	51.0s
$\omega_3^{(3a)}$	1.80s	2.45s
$\omega_1^{(3b)}$	6.53s	0.825s

Table 6.3.6

Note: When using $\omega_3^{(3a)}$ to prove $\mathcal{A}_{3b}^{LOG}$ ($\omega_1^{(3b)}$ to prove $\mathcal{A}_{3a}^{LOG}$), the roles of u and l were again reversed. But this is not necessary here. Without reversing the roles, $\omega_3^{(3a)}$ ($\omega_1^{(3b)}$) allows for a proof of $\mathcal{A}_{3b}^{LOG}$ ($\mathcal{A}_{3a}^{LOG}$) within 2.2 (6.1) seconds. There is hence no notable difference between both alternatives.

The main aspect of this example is the fact that the "antisymmetry" of $\mathcal{A}_{3a}^{LOG}$ and $\mathcal{A}_{3b}^{LOG}$ (w.r.t. exchanging u and l) is not only reflected by the findings described in the above note, but also by an "antisymmetric learning behavior": $\omega_3^{(3a)}$, which learned the proof of $\mathcal{A}_{3a}^{LOG}$, is equally profitable for $\mathcal{A}_{3a}^{LOG}$ and for $\mathcal{A}_{3b}^{LOG}$, whereas $\omega_1^{(3b)}$ is not, showing merely moderate improvement w.r.t. $\mathcal{A}_{3a}^{LOG}$. This phenomenon deserves a closer investigation beyond the focus of this report.

6.4 The fourth set of examples: Completion tasks

The fourth and final set of examples comprises eleven completion tasks $\mathcal{A}_8$, $\mathcal{A}_{10}$, $\mathcal{A}_{20}$, $\mathcal{A}_{30}$, $\mathcal{A}_{40}$, $\mathcal{A}_{50}$, $\mathcal{A}_{60}$, $\mathcal{A}_{70}$, $\mathcal{A}_{80}$, $\mathcal{A}_{90}$ and $\mathcal{A}_{100}$ (cp. [Ch93], [Zh92]).

[6]Only *occnest* could learn the proof of $\mathcal{A}_{3a}^{LOG}$ resp. $\mathcal{A}_{3b}^{LOG}$ and exhibit significant improvements. (*linpol* was not considered for reasons explained in the introduction of this section.)

The initial set of equations for $\mathcal{A}_n$ is

$$\begin{array}{ll} f(e_j, x) = x \quad , & 1 \leq j \leq n \\ f(x, i_j(x)) = e_j \quad , & 1 \leq j \leq n \\ f(f(x,y),z) = f(x, f(y,z)). & \end{array}$$

The completed system has $3n + 6$ rules and no equations if the reduction ordering is a LPO with precedence $i_n \succ_p \cdots \succ_p i_1 \succ_p f \succ_p e_n \succ_p \cdots \succ_p e_1$.

Note that it still makes sense to talk about proofs in this context if we view the completion process as proving the rules and equations of the resulting convergent system.

The first three rows of table 6.4.1 list the results of the default heuristics *add*, *max* and *gt*. (We omitted *occnest* because it does not make sense to use a goal oriented heuristic for completion.) The entry '∞' indicates that no convergent system could be produced, because DISCOUNT ran out of memory. The fourth row displays the results produced by the best adapted heuristic ω_2 (cp. table B.4.1) which was generated by adapting $\mathcal{W} \equiv max$ to the completion of the *simplest* task $\mathcal{A}_8$. Here we set $T = 3$ seconds. The fifth row shows the results produced by Herky, which is one of the most powerful equational provers currently available (cf. [Zh92], [Zh93]). Note that Herky is implemented in LISP, but can nevertheless defy serious C implementations (e.g., OTTER) w.r.t. many equational problems.

	$\mathcal{A}_8$	$\mathcal{A}_{10}$	$\mathcal{A}_{20}$	$\mathcal{A}_{30}$	$\mathcal{A}_{40}$	$\mathcal{A}_{50}$	$\mathcal{A}_{60}$	$\mathcal{A}_{70}$	$\mathcal{A}_{80}$	$\mathcal{A}_{90}$	$\mathcal{A}_{100}$
add	1.6s	2.9s	27.0s	110s	340s	850s	∞	∞	∞	∞	∞
max	3.9s	8.0s	93.8s	1120s	4600s	∞	∞	∞	∞	∞	∞
gt	3.9s	8.0s	99.9s	1141s	4630s	∞	∞	∞	∞	∞	∞
ω_2	0.23s	0.28s	0.69s	1.37s	2.16s	3.18s	4.16s	5.34s	6.87s	8.56s	10.1s
Herky	4.2s	6.8s	25.0s	79.9s	179s	323s	623s	1031s	1610s	2238s	2964s

Table 6.4.1: *default versions, best adapted heuristic (max) and* Herky

Again, the performance of DISCOUNT could be enhanced significantly. To underline the achievements of our method we refer to [Zh92] where the automated prover Herky is compared to OTTER (probably an earlier release than version 3.0) and Hiper ([Ch93]). Herky outperforms both OTTER and Hiper, because it generates far less critical pairs due to the application of sophisticated, but time consuming techniques. But the (run-times and) number of generated critical pairs reported in [Zh92] are no match to our results. For instance, when employing ω_2 DISCOUNT generates 11,336 critical pairs during the completion of $\mathcal{A}_{100}$, whereas Herky generates 73,144. Moreover, all these improvements stem from the effort of learning made *once* (about 3 minutes learning time), which starts to pay off beyond $\mathcal{A}_{40}$ (w.r.t. Herky, earlier w.r.t. *add*, *max* or *gt*), even if *merely one* application of the learned heuristic is taken into account.

Technical remark: The problem that $\mathcal{A}_i$ has a different number of function symbols than $\mathcal{A}_j$ for $i \neq j$ was resolved in the following way: Assuming that $\mathcal{A}_i$ is the completion task $\mathcal{W}$ was adapted to resulting in ω, three cases arise when transforming ω into ω' then to be used for the completion of $\mathcal{A}_j$.

(i)	$i > j$	Construct ω' by simply omitting from ω the function symbols $e_{j+1}, \ldots, e_i$ and $i_{j+1}, \ldots, i_i$ not occurring in $\mathcal{A}_j$.
(ii)	$i = j$	Use ω as it is, i.e., $\omega' \equiv \omega$.
(iii)	$i < j$	ω' is obtained by treating the additional $e_{i+1}, \ldots, e_j$ $(i_{i+1}, \ldots, i_j)$ like the average $e_1, \ldots, e_i$ $(i_1, \ldots, i_i)$ in ω, precluding those e_k (i_k), $1 \leq i \leq k$, that deviate considerably from the average. That means that the weights of the new function symbols become the "average" weight of the "corresponding" known function symbols. This way of proceeding is of course only a heuristic motivated by the intuitively convincing hypothesis that additional e_k or i_k will play the same role as the majority of the e_m and i_m already there.

(*Remark*: Adding or omitting function symbols was done manually. But this procedure will not be difficult to automate as long as the problems are as transparently structured as they are here.)

7 Related work

Poor performance of automated proving systems when it comes to proving difficult theorems, and the resulting lack of competitiveness compared to mathematicians are the main reasons to improve the heuristics of the respective provers. Although most designers of automated provers have recognized that the best and most natural way to do this is by learning from previous successfully solved tasks (e.g., [BCP88], [Bu88] and more recently [KW94]) the number of reports on *automated* approaches to learning in this environment *substantiated by experimental results* is rather low. To our knowledge, there has so far no work been done aiming at integrating an automated learning component into a prover for purely equational logic. We are aware of two research papers dealing with such a component for provers for first order logic ([SF71] and [SE90]). Common to both approaches is the representation of knowledge as clauses (CNF), and both use so-called *features* of clauses (see also [Su90]) as the basis of learning.

Features reflect certain properties of clauses, e.g., the total number of literals, the number of positive and negative literals etc. In [SF71], *feature vectors* are extracted from the data provided by a successful proof, each feature vector belonging to one clause derived during search. "Profit values" are associated with each feature vector, each expressing in some respect the usefulness of the related clause. Learning consists in finding functions approximating the 'feature vector vs. profit value'-relation. These functions are linear polynoms (in the features) whose coefficients are determined via multiple regression analysis (see [SF71] for details).

[SE90] also use feature vectors which are—properly encoded—presented to a neural network. The desired input–output behavior of the net is extracted from successful proofs (The only output unit is supposed to produce 1 for "useful" clauses, 0 for "useless" ones).

Both approaches will not work when applied to pure equational deduction unless additional features are introduced. Most of the current features are not distinctive when the clauses are limited to unit clauses with the equality predicate as the only predicate.

We believe that our approach is not limited to (pure) equational deduction, but can be applied to any search problem that is tackled using parameterized heuristics. The quality of the results will of course heavily depend on the adequacy of these heuristics. Furthermore, our approach causes no *additional* overhead when a learned heuristic is used during a proof run. Both [SF71] and [SE90] have to accept overhead not present in the original prover due to additional computations caused by feature evaluation and—only for [SE90]—processing the resulting data through a neural net.

In the general context of proof reuse also the work reported in [KW94] ought to be mentioned. The approach taken there combines ideas from explanation based learning and analogical reasoning as well as abstraction techniques. It consequently differs substantially from our approach, since proof reuse is accomplished by analyzing and generalizing proofs found in the past, which are then—properly instantiated—utilized for finding similar proofs. In [KW94], the notion 'similarity' is clearly defined, but proofs have to be very similar in this sense in order for this approach to be successful.

8 Summary

We have shown for a prover for purely equational logic that by adapting (learning) the parameters of a proof heuristic $\mathcal{W}$ via a genetic algorithm not only the proof that $\mathcal{W}$ was adapted to can be found considerably faster, but also the reduction of redundancies carries over to proofs that are in some way "similar". We emphasize that learning and application are completely independent of each other. This entails that the effort made for learning has to be made only once, and there are no additional costs involved when applying an adapted heuristic. We believe that our approach is not limited to (purely) equational deduction, but can be applied to any deductive system that tackles problems with parameterized heuristics and explicitly infers facts.

We have conducted numerous experiments demonstrating the capabilities of our approach to reusing proofs. We hope to have illustrated with their results summarized in section 6 that our method is naturally not *the* answer to "reusing proofs", but is quite a powerful member of a future set of methods dealing with this problematic.

Finally, this report deals with the problematic *how* to learn and this way (indirectly) reuse a known proof. Future work has to concentrate on an inherent difficulty of proof reuse not addressed in this report, namely to determine *when* to apply experiences gained in the past. A judicious decision on that score is crucial in order to avoid a major pitfall of proof reuse which consists in sometimes causing a change for the worse compared to proving from scratch (cp. [KN93]).

A The heuristic "occnest"

The heuristic *occnest* is a goal oriented selection strategy for critical pairs based on so-called measures which were first introduced by S. Anantharaman and N. Andrianarievelo ([AA90]). Measures express some property of a term t (e.g., the number of occurrences of a function symbol f in t) as integer values. Goal orientation is achieved by comparing measures acquired from a critical pair $\langle u, v\rangle$ with the corresponding measures obtained from the goal $s \neq t$. The comparison can be accomplished in many ways. The way we chose is sketched below. For a more detailed discussion see [Fu94], [DF94].

We already know (cf. definition 4.10) that *occnest* is defined as follows:

$$occnest_{s\neq t}(\langle u, v\rangle) = (\phi(u) + \phi(v)) \cdot \prod_{f\in\mathcal{F}} m_f$$

where ϕ is the weighting function for terms from definition 4.8. The multipliers m_f ($f \in \mathcal{F}$) can be computed by comparing the measures of $\langle u, v\rangle$ w.r.t. f with the respective measures of $s \neq t$ (w.r.t. f). Two measures are employed for *occnest*, namely **occurrences** and **nesting**. We shall first define both notions for terms and then extend these definitions to pairs of terms, which is necessary since *occnest* operates on (two) pairs of terms (a critical pair $\langle u, v\rangle$ and the goal $s \neq t$).

Definition A.1 (occurrences) *Let $f \in \mathcal{F}$, $t \in Term(\mathcal{F}, \mathcal{V})$. The (number of)* **occurrences** *of f in t is defined by*

$$occ(f,t) = \begin{cases} 0, & if\ t \in \mathcal{V} \\ \sum_{i=1}^{n} occ(f,t_i), & if\ t \equiv g(t_1,\ldots,t_n),\ g \not\equiv f \\ 1 + \sum_{i=1}^{n} occ(f,t_i), & if\ t \equiv f(t_1,\ldots,t_n) \end{cases}$$

The nesting of f in t as given by the following definition captures the maximal number of consecutive occurrences of f on the branches of t (t is viewed as a tree).

Definition A.2 (nesting) *Let $f \in \mathcal{F}$, $t \in Term(\mathcal{F}, \mathcal{V})$. The* **nesting** *of f in t is defined by*

$$nest(f,t) = \begin{cases} 0, & if\ f\ is\ a\ constant \\ \Upsilon(f,t,0,0), & otherwise \end{cases}$$

where

$$\Upsilon(f,t,c,a) = \begin{cases} \max(\{c,a\}), & if\ t \in \mathcal{V}\ or\ t\ is\ a\ constant \\ \max(\{\Upsilon(f,t_i,0,\max(\{c,a\})) \mid 1 \le i \le n\}), & if\ t \equiv g(t_1,\ldots,t_n),\ g \not\equiv f \\ \max(\{\Upsilon(f,t_i,c+1,a) \mid 1 \le i \le n\}) & if\ t \equiv f(t_1,\ldots,t_n) \end{cases}$$

occ and *nest* are made applicable to pairs of terms in the following way:

$$\begin{aligned} Occ(f,\langle u,v\rangle) &= \max(\{occ(f,u), occ(f,v)\}) \\ Nest(f,\langle u,v\rangle) &= \max(\{nest(f,u), nest(f,v)\}) \end{aligned}$$

Now we can define the multipliers:

Definition A.3 (multipliers) *Let $\mathcal{D} \subseteq \mathcal{F}$, $\langle u, v\rangle$ a critical pair, $s \neq t$ the goal (i.e., $u, v \in Term(\mathcal{F}, \mathcal{V})$, $s, t \in Term(\mathcal{F} \cup \mathcal{F}_{sk}, \mathcal{V})$, where $\mathcal{F}_{sk}$ denotes the set of Skolem symbols). For all $f \in \mathcal{F}$:*

$$m_f = \begin{cases} 1, & \textit{if } f \notin \mathcal{D} \\ \theta(Occ(f, \langle u,v\rangle) - Occ(f, \langle s,t\rangle)) \cdot \theta(Nest(f, \langle u,v\rangle) - Nest(f, \langle s,t\rangle)), & \textit{else} \end{cases}$$

where

$$\theta(x) = \begin{cases} x+1, & \textit{if } x > 0 \\ 1, & \textit{otherwise} \end{cases}$$

$\mathcal{D}$ as a subset of $\mathcal{F}$ allows to preclude certain function symbols (namely those not in $\mathcal{D}$). The measures of a function symbol $f \notin \mathcal{D}$ are simply ignored ($m_f = 1$). When using $\mathcal{D} = \emptyset$ *occnest* degenerates into the heuristic *add*.

In our implementation we use a flag admitting either $\mathcal{D} = \mathcal{F}$ or $\mathcal{D} = \{ f \in \mathcal{F} \mid Occ(f, \langle s,t\rangle) > 0 \}$, i.e., the set of all function symbols occurring in the goal $s \neq t$. Furthermore, note that we always have $m_f \geq 1$, and m_f increases the more (at least one of) these measures of a critical pair exceed the thresholds set by the goal (provided that $f \in \mathcal{D}$).

Experiments with the default versions (cp. section 6) showed that *occnest* is in many cases substantially superior to the heuristic *add* (*linpol*), *max* and *gt*[7] (see [Fu94]).

[7] We must point out that we used a different version of *gt* in [Fu94]. Although the version used now is often superior to the former version, the fact that *gt* is still outperformed by *occnest* is lasting.

B Complete listing of all results

We present here the complete listing of all experimental results which are referenced by section 6. Each of the following tables shows all the "successful" instances ω of a heuristic $\mathcal{W}$ generated by the adaptive procedure during ten iterations. $\mathcal{W}$ is specified at the bottom of each table. The proof problem $\mathcal{A}$ (resp. the proof of $\mathcal{A}$) it was adapted to is given in the left upper corner of each table.

This appendix is divided in subsections the same way section 6 is. Therefore the tables in subsections B.1, B.2, B.3 and B.4 refer to the problems dealt with in subsections 6.1, 6.2, 6.3 and 6.4, respectively.

In any of the tables below the indices of the instances ω do not necessarily correspond to the iteration the respective ω arose from. But they do reflect the chronological order.

Please note that the ω_i of a table do not always improve monotonously (with the number of iterations). On the contrary, they sometimes exhibit a rather irregular behavior. This property, as well as occasional "setbacks", are caused by the sudden changes of the fitness function on account of update runs.

An entry '—' signifies that no proof could be found in a time comparable to that of a default heuristic or another adapted heuristic.

B.1 Non-associative rings

$\mathcal{A}_1^{NA}$	$\mathcal{A}_1^{NA}$	$\mathcal{A}_2^{NA}$	$\mathcal{A}_3^{NA}$
ω_1	2.072s	2.076s	2.064s
ω_2	0.796s	0.803s	0.743s
ω_3	0.337s	0.342s	0.337s

Table B.1.1: $\mathcal{W} \equiv gt$

$\mathcal{A}_2^{NA}$	$\mathcal{A}_1^{NA}$	$\mathcal{A}_2^{NA}$	$\mathcal{A}_3^{NA}$
ω_1	2.781s	2.817s	2.793s
ω_2	0.341s	0.349s	0.342s
ω_3	0.344s	0.350s	0.353s

Table B.1.2: $\mathcal{W} \equiv gt$

$\mathcal{A}_3^{NA}$	$\mathcal{A}_1^{NA}$	$\mathcal{A}_2^{NA}$	$\mathcal{A}_3^{NA}$
ω_1	2.063s	2.079s	2.067s
ω_2	0.795s	0.803s	0.735s
ω_3	0.336s	0.345s	0.337s

Table B.1.3: $\mathcal{W} \equiv gt$

B.2 Propositional logic

$\mathcal{A}_1^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	7.9s	26.2s	8.4s	7.8s	16.7s	16.4s	9.2s	14.6s	11.6s	18.2s
ω_2	11.5s	27.7s	7.8s	10.8s	17.5s	17.2s	9.8s	15.2s	11.6s	18.3s

Table B.2.2.1: $\mathcal{W} \equiv occnest$

$\mathcal{A}_2^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	2.3s	—	—	2.1s	2.5s	—	—	13.0s	3.6s
ω_2	—	2.9s	—	—	—	—	—	—	—	—
ω_3	—	2.3s	—	—	2.2s	1.9s	—	—	15.3s	3.7s
ω_4	—	2.4s	—	—	1.8s	2.2s	—	—	15.3s	3.5s
ω_5	—	2.7s	—	—	—	—	—	—	—	—
ω_6	—	2.3s	—	—	1.8s	1.9s	—	—	15.2s	3.6s
ω_7	—	2.2s	—	—	1.8s	1.9s	—	—	15.3s	3.6s
ω_8	—	2.2s	—	—	1.8s	1.9s	—	—	15.3s	3.5s
ω_9	—	2.8s	—	—	—	—	—	—	—	—

Table B.2.2.2: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_3^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	—	4.9s	—	—	—	9.4s	15.6s	10.6s	—
ω_2	—	—	4.4s	—	—	—	9.5s	15.4s	10.8s	—
ω_3	—	—	4.4s	—	—	—	9.0s	15.5s	10.9s	—

Table B.2.2.3: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_4^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	7.9s	29.3s	7.8s	7.7s	16.5s	16.5s	9.1s	14.7s	11.1s	18.2s
ω_2	7.8s	26.3s	7.8s	7.8s	16.6s	16.5s	9.2s	14.7s	11.1s	18.2s
ω_3	7.8s	28.2s	10.3s	7.9s	16.5s	17.1s	9.4s	15.9s	12.4s	21.6s

Table B.2.2.4: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_5^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	12.2s	—	—	2.1s	2.2s	—	—	2.0s	3.5s
ω_2	—	2.2s	—	—	1.8s	1.9s	—	—	15.3s	3.5s
ω_3	—	2.4s	—	—	1.8s	1.9s	—	—	15.3s	3.5s
ω_4	—	11.8s	—	—	2.1s	2.2s	—	—	2.4s	3.5s
ω_5	—	2.3s	—	—	2.0s	2.1s	—	—	15.2s	2.5s

Table B.2.2.5: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_6^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	12.7s	—	—	2.1s	2.2s	—	—	2.1s	3.5s
ω_2	—	2.3s	—	—	1.8s	1.9s	—	—	18.6s	3.5s
ω_3	—	2.2s	—	—	1.8s	1.9s	—	—	15.2s	3.6s
ω_4	—	2.3s	—	—	2.0s	2.0s	—	—	15.3s	3.5s
ω_5	—	2.2s	—	—	1.8s	1.9s	—	—	12.8s	2.5s
ω_6	—	2.3s	—	—	1.8s	1.9s	—	—	15.3s	3.5s

Table B.2.2.6: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_7^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	—	18.6s	—	28.2s	28.0s	4.1s	5.7s	2.7s	24.5s
ω_2	—	19.9s	—	—	30.0s	29.8s	2.7s	3.8s	2.4s	—

Table B.2.2.7: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_8^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	17.7s	—	—	27.7s	28.0s	7.4s	7.7s	1.7s	21.1s
ω_2	—	—	18.5s	—	28.4s	28.2s	4.1s	5.0s	2.5s	24.6s
ω_3	—	19.0s	—	—	31.4s	31.2s	3.0s	4.3s	2.5s	—
ω_4	—	—	7.7s	—	26.0s	25.8s	5.5s	6.9s	6.0s	23.4s
ω_5	—	20.1s	—	—	30.4s	30.2s	2.7s	3.8s	2.8s	—
ω_6	—	19.8s	—	—	30.1s	29.9s	3.0s	4.2s	2.3s	—
ω_7	—	—	7.7s	—	26.4s	26.0s	5.4s	6.9s	6.0s	21.5s

Table B.2.2.8: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_9^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	19.6s	—	—	30.1s	29.9s	3.0s	4.2s	2.3s	—
ω_2	—	11.9s	—	—	27.7s	28.0s	17.9s	19.2s	1.7s	21.1s
ω_3	—	20.1s	—	—	30.1s	29.9s	2.7s	3.8s	2.3s	—
ω_4	—	17.0s	—	—	27.9s	28.2s	17.5s	18.9s	1.5s	21.1s
ω_5	—	17.2s	—	—	27.6s	27.9s	17.6s	18.8s	1.5s	21.1s
ω_6	—	22.4s	—	—	30.9s	30.7s	3.7s	4.6s	2.4s	—

Table B.2.2.9: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_{10}^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	12.3s	—	—	3.2s	3.3s	17.9s	18.7s	2.5s	2.8s
ω_2	—	2.4s	—	—	2.0s	2.1s	—	—	13.3s	2.6s
ω_3	—	2.4s	—	—	2.0s	2.1s	—	—	13.3s	2.6s
ω_4	—	2.4s	—	—	2.1s	2.2s	—	—	13.5s	2.7s

Table B.2.2.10: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_1^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	4.9s	5.4s	1.3s	4.8s	3.1s	3.2s	—	—	21.3s	3.2s

Table B.2.3.1: $\mathcal{W} \equiv$ *linpol*

$\mathcal{A}_2^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	29.1s	4.3s	33.4s	28.9s	0.8s	0.9s	—	—	—	0.9s

Table B.2.3.2: $\mathcal{W} \equiv$ *linpol*

$\mathcal{A}_3^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	24.2s	27.1s	1.7s	24.0s	15.4s	15.7s	2.0s	1.7s	1.8s	18.8s
ω_2	—	—	0.7s	—	—	—	0.7s	0.7s	0.7s	—
ω_3	—	—	0.9s	—	26.4s	26.7s	1.0s	1.0s	1.0s	26.7s
ω_4	—	—	1.2s	—	—	—	1.2s	1.2s	3.4s	—
ω_5	—	—	1.0s	—	—	—	1.0s	1.0s	1.2s	—
ω_6	—	—	0.8s	—	25.7s	25.0s	0.9s	0.9s	2.8s	24.8s

Table B.2.3.3: $\mathcal{W} \equiv linpol$

$\mathcal{A}_4^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	4.9s	5.4s	1.3s	4.8s	3.2s	3.3s	—	—	—	3.2s

Table B.2.3.4: $\mathcal{W} \equiv linpol$

$\mathcal{A}_5^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	—	—	—	1.0s	1.2s	—	—	—	0.9s
ω_2	20.0s	29.9s	24.2s	20.5s	1.0s	1.6s	—	—	—	1.0s
ω_3	—	—	—	—	0.6s	0.9s	—	—	—	0.6s
ω_4	—	—	—	—	0.6s	1.0s	—	—	—	0.6s
ω_5	—	—	—	—	0.5s	0.5s	—	—	—	0.5s
ω_6	—	—	—	—	0.6s	0.7s	—	—	—	0.5s
ω_7	—	—	—	—	0.8s	1.0s	—	—	—	0.7s

Table B.2.3.5: $\mathcal{W} \equiv linpol$

$\mathcal{A}_6^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	—	—	—	1.7s	1.8s	—	—	—	1.8s
ω_2	—	—	—	—	1.5s	1.6s	—	—	—	1.6s
ω_3	—	—	—	—	1.2s	1.3s	—	—	—	1.3s
ω_4	—	—	—	—	2.1s	2.2s	—	—	—	2.2s
ω_5	—	—	—	—	1.3s	1.4s	—	—	—	1.4s
ω_6	—	—	—	—	1.2s	1.4s	—	—	—	1.5s
ω_7	6.9s	6.2s	9.1s	6.9s	1.9s	1.9s	—	—	—	2.0s

Table B.2.3.6: $\mathcal{W} \equiv linpol$

$\mathcal{A}_7^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	24.2s	1.6s	—	9.3s	9.5s	1.6s	1.6s	1.6s	9.5s
ω_2	—	—	1.3s	—	—	—	1.4s	1.4s	1.4s	—
ω_3	—	—	1.0s	—	—	—	1.0s	1.0s	1.0s	—
ω_4	—	—	2.0s	—	—	—	2.4s	2.4s	2.4s	—
ω_5	—	—	0.6s	—	20.2s	20.5s	0.6s	0.6s	0.6s	20.3s
ω_6	—	30.9s	0.8s	—	19.5s	19.9s	0.8s	0.8s	0.8s	19.9s
ω_7	—	—	0.6s	—	—	—	0.6s	0.6s	0.6s	—

Table B.2.3.7: $\mathcal{W} \equiv linpol$

$\mathcal{A}_8^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	13.8s	1.3s	—	8.5s	8.6s	1.3s	1.3s	1.3s	8.7s
ω_2	—	—	1.0s	—	—	—	1.0s	1.0s	1.0s	—
ω_3	—	—	0.6s	—	—	—	0.6s	0.6s	0.6s	—
ω_4	9.8s	11.0s	0.8s	9.6s	5.2s	5.3s	0.9s	0.8s	0.8s	5.4s
ω_5	—	—	0.9s	—	—	—	1.0s	1.0s	1.0s	—
ω_6	—	—	1.6s	—	13.4s	13.5s	1.6s	1.6s	1.6s	13.5s

Table B.2.3.8: $\mathcal{W} \equiv linpol$

$\mathcal{A}_9^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	—	0.9s	—	—	—	1.0s	1.0s	1.0s	—
ω_2	—	21.0s	2.0s	—	14.3s	14.5s	2.4s	2.4s	2.4s	14.5s
ω_3	—	—	0.9s	—	—	—	1.0s	1.0s	1.0s	—
ω_4	—	—	1.1s	—	28.7s	29.1s	1.2s	1.2s	1.2s	28.9s
ω_5	—	—	1.0s	—	—	—	1.0s	1.0s	1.0s	—
ω_6	—	—	0.7s	—	—	—	0.7s	0.7s	0.7s	—
ω_7	—	—	0.6s	—	16.5s	16.6s	0.6s	0.6s	0.7s	16.6s

Table B.2.3.9: $\mathcal{W} \equiv linpol$

$\mathcal{A}_{10}^{PL}$	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$	$\mathcal{A}_9^{PL}$	$\mathcal{A}_{10}^{PL}$
ω_1	—	—	—	—	1.5s	1.5s	—	—	—	1.5s
ω_2	5.7s	7.5s	5.0s	5.6s	2.3s	2.4s	—	—	—	2.4s
ω_3	—	—	—	—	2.7s	2.8s	—	—	—	2.8s
ω_4	—	—	—	—	0.7s	0.7s	—	—	—	0.7s
ω_5	39.1s	—	—	38.9s	0.8s	0.9s	—	—	—	0.8s
ω_6	16.7s	—	7.4s	16.9s	4.7s	4.8s	—	—	—	4.8s

Table B.2.3.10: $\mathcal{W} \equiv linpol$

B.3 Lattice ordered groups

$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
ω_1	0.900s	1.018s	0.824s	0.829s
ω_2	0.795s	0.802s	0.993s	1.284s
ω_3	3.277s	8.356s	—	—
ω_4	1.704s	2.475s	0.462s	0.585s
ω_5	0.227s	1.040s	0.236s	0.296s
ω_6	1.335s	1.448s	1.494s	1.682s
ω_7	0.294s	0.295s	0.370s	0.399s
ω_8	0.350s	0.350s	0.744s	0.727s

Table B.3.2.1a: $\mathcal{W} \equiv add$

$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{1a}^{LOG}$	$\mathcal{A}_{1b}^{LOG}$	$\mathcal{A}_{2a}^{LOG}$	$\mathcal{A}_{2b}^{LOG}$
ω_1	3.602s	0.210s	0.215s	—
ω_2	2.675s	2.252s	1.425s	3.609s
ω_3	0.432s	0.427s	0.530s	0.662s
ω_4	1.354s	4.445s	1.447s	2.829s
ω_5	100.5s	77.9s	2.119s	—
ω_6	0.537s	0.495s	0.564s	0.575s
ω_7	1.270s	3.124s	0.259s	0.538s
ω_8	0.303s	0.300s	0.324s	0.498s

Table B.3.2.2a: $\mathcal{W} \equiv add$

$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.253s	0.236s	0.251s	0.222s
ω_2	0.376s	0.381s	0.426s	0.420s
ω_3	0.309s	0.283s	0.453s	0.364s
ω_4	0.635s	0.789s	0.735s	0.958s
ω_5	36.9s	0.267s	98.9s	0.257s
ω_6	1.186s	0.271s	0.323s	0.264s
ω_7	0.271s	0.267s	0.378s	0.422s

Table B.3.2.1b: $\mathcal{W} \equiv add$

$\mathcal{A}^{LOG}_{2b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	7.209s	4.167s	0.456s	0.270s
ω_2	0.538s	0.637s	0.638s	0.855s
ω_3	38.9	21.0	109.9s	1.516s
ω_4	1.743s	0.361s	0.550s	0.388s
ω_5	0.827s	0.800s	2.542s	2.909s

Table B.3.2.2b: $\mathcal{W} \equiv add$

$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	4.584s	1.326s	29.2s	50.6s
ω_2	0.553s	0.729s	0.422s	0.339s
ω_3	0.173s	0.185s	0.212s	0.223s
ω_4	0.191s	0.196s	0.232s	0.218s
ω_5	1.916s	1.891s	0.338s	0.336s
ω_6	0.166s	0.174s	0.226s	0.209s
ω_7	0.193s	0.196s	0.220s	0.219s

Table B.3.3.1a: $\mathcal{W} \equiv max$

$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.198s	0.197s	0.226s	0.211s
ω_2	0.193s	0.265s	0.218s	0.215s
ω_3	1.831s	3.635s	1.967s	3.424s
ω_4	10.5s	18.4s	0.250s	0.238s
ω_5	3.523s	5.377s	0.232s	0.211s
ω_6	2.804s	4.914s	0.235s	0.223s
ω_7	1.843s	3.253s	0.209s	0.195s
ω_8	1.878s	3.252s	0.235s	0.223s

Table B.3.3.2a: $\mathcal{W} \equiv max$

$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.187s	0.189s	0.208s	0.210s
ω_2	1.992s	2.010s	117.6s	118.5s
ω_3	0.179s	0.190s	0.224s	0.220s
ω_4	0.255s	0.276s	0.287s	0.282s
ω_5	0.191s	0.199s	0.220s	0.233s
ω_6	1.895s	1.990s	0.328s	0.219s
ω_7	1.890s	1.953s	0.241s	0.234s

Table B.3.3.1b: $\mathcal{W} \equiv max$

$\mathcal{A}^{LOG}_{2b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.204s	0.203s	0.218s	0.215s
ω_2	0.797s	0.653s	2.073s	1.995s
ω_3	5.437s	5.589s	0.213s	0.206s
ω_4	3.337s	3.516s	0.219s	0.198s
ω_5	3.599s	3.746s	0.211s	0.210s
ω_6	2.064s	2.164s	0.241s	0.242s
ω_7	1.695s	1.791s	0.218s	0.207s
ω_8	0.756s	0.915s	0.208s	0.211s

Table B.3.3.2b: $\mathcal{W} \equiv max$

$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	3.066s	3.147s	3.088s	3.085s
ω_2	0.732s	0.791s	2.410s	4.537s
ω_3	0.381s	0.367s	0.215s	0.229s
ω_4	0.857s	0.878s	0.635s	0.668s
ω_5	0.643s	0.700s	0.304s	0.316s
ω_6	3.007s	2.586s	2.541s	11.8s
ω_7	1.828s	2.278s	2.006s	1.988s
ω_8	2.128s	2.079s	1.028s	1.443s

Table B.3.4.1a: $\mathcal{W} \equiv gt$

$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	6.029s	10.3s	2.599s	2.516s
ω_2	2.733s	2.654s	1.576s	1.611s
ω_3	2.564s	2.550s	2.289s	2.420s
ω_4	1.781s	1.850s	0.873s	0.914s
ω_5	1.944s	2.490s	0.580s	0.627s
ω_6	1.781s	1.952s	0.872s	0.885s
ω_7	1.174s	1.248s	0.990s	1.042s

Table B.3.4.2a: $\mathcal{W} \equiv gt$

$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.605s	0.609s	0.259s	0.332s
ω_2	0.442s	0.417s	0.454s	0.469s
ω_3	0.295s	0.274s	0.447s	0.553s
ω_4	1.113s	1.289s	0.878s	1.409s
ω_5	0.321s	0.298s	0.303s	0.325s
ω_6	0.339s	0.327s	0.440s	0.644s
ω_7	0.338s	0.379s	0.439s	0.472s

Table B.3.4.1b: $\mathcal{W} \equiv gt$

$\mathcal{A}^{LOG}_{2b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	1.772s	1.880s	1.659s	1.673s
ω_2	2.023s	2.064s	2.328s	2.427s
ω_3	1.909s	1.953s	1.188s	1.187s
ω_4	2.343s	2.450s	2.074s	2.160s
ω_5	1.289s	1.337s	1.652s	2.084s

Table B.3.4.2b: $\mathcal{W} \equiv gt$

$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.214s	0.292s	0.275s	0.226s
ω_2	0.909s	1.785s	1.792s	3.038s
ω_3	0.321s	0.341s	0.718s	1.102s
ω_4	0.434s	3.555s	0.219s	0.294s
ω_5	0.442s	0.415s	0.443s	0.899s
ω_6	0.463s	0.412s	1.376s	1.207s
ω_7	0.586s	0.414s	1.182s	1.279s
ω_8	0.433s	0.612s	1.173s	1.309s
ω_9	0.410s	0.395s	2.051s	2.070s
ω_{10}	0.170s	0.138s	1.012s	1.007s

Table B.3.5.1a: $\mathcal{W} \equiv occnest$

$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	8.303s	0.240s	0.284s	7.521s
ω_2	0.207s	0.220s	0.333s	0.324s
ω_3	0.447s	11.8s	0.670s	1.161s
ω_4	—	—	2.520s	5.076s
ω_5	2.323s	4.021s	0.411s	9.974s
ω_6	2.472s	2.426s	0.467s	8.556s
ω_7	26.2s	1.659s	0.504s	0.591s
ω_8	9.958s	5.431s	2.052s	1.301s
ω_9	0.413s	9.344s	0.455s	0.673s
ω_{10}	15.5s	1.692s	0.522s	0.564s

Table B.3.5.2a: $\mathcal{W} \equiv occnest$

$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	0.210s	0.231s	0.318s	0.561s
ω_2	1.635s	1.302s	2.876s	2.844s
ω_3	0.320s	0.310s	0.370s	0.448s
ω_4	1.554s	0.213s	0.442s	0.398s
ω_5	0.427s	0.446s	1.755s	1.744s
ω_6	0.433s	0.451s	2.355s	2.492s
ω_7	0.424s	0.443s	2.618s	2.253s
ω_8	0.420s	0.446s	2.423s	2.556s
ω_9	0.263s	0.305s	0.442s	0.491s

Table B.3.5.1b: $\mathcal{W} \equiv occnest$

$\mathcal{A}^{LOG}_{2b}$	$\mathcal{A}^{LOG}_{1a}$	$\mathcal{A}^{LOG}_{1b}$	$\mathcal{A}^{LOG}_{2a}$	$\mathcal{A}^{LOG}_{2b}$
ω_1	—	—	5.267s	3.853s
ω_2	0.450s	1.609s	11.7 s	0.226s
ω_3	14.2s	1.441s	2.901s	3.173s
ω_4	1.758s	0.421s	0.733s	0.439s
ω_5	2.300s	2.307s	0.418s	0.420s
ω_6	14.2s	2.920s	1.547s	1.342s
ω_7	1.748s	0.391s	0.558s	0.407s
ω_8	2.824s	3.000s	0.349s	0.349s
ω_9	3.269s	0.326s	0.509s	0.474s

Table B.3.5.2b: $\mathcal{W} \equiv occnest$

$\mathcal{A}_{3a}^{LOG}$	$\mathcal{A}_{3a}^{LOG}$	$\mathcal{A}_{3b}^{LOG}$	$\mathcal{A}_{3b}^{LOG*}$
ω_1	3.006s	2.441s	1.347s
ω_2	3.646s	2.516s	1.836s
ω_3	1.801s	2.449s	2.138s

Table B.3.6.3a: $\mathcal{W} \equiv$ *occnest*

$\mathcal{A}_{3b}^{LOG}$	$\mathcal{A}_{3a}^{LOG}$	$\mathcal{A}_{3b}^{LOG}$	$\mathcal{A}_{3a}^{LOG*}$
ω_1	6.530s	0.825s	6.108s
ω_2	—	3.992s	—
ω_3	10.9s	1.170s	21.9s
ω_4	29.3s	2.004s	—
ω_5	14.8s	0.835s	55.6s
ω_6	28.0s	2.216s	—
ω_7	13.9s	2.013s	—
ω_8	14.6s	2.102s	—
ω_9	33.9s	3.337s	—

Table B.3.6.3b: $\mathcal{W} \equiv$ *occnest*

In the columns marked with an asterisk the roles of u and l were not reversed (cp. subsection 6.3).

B.4 Completion tasks

$\mathcal{A}_8$	$\mathcal{A}_8$	$\mathcal{A}_{10}$	$\mathcal{A}_{20}$	$\mathcal{A}_{30}$	$\mathcal{A}_{40}$	$\mathcal{A}_{50}$	$\mathcal{A}_{60}$	$\mathcal{A}_{70}$	$\mathcal{A}_{80}$	$\mathcal{A}_{90}$	$\mathcal{A}_{100}$
ω_1	0.32s	0.38s	0.88s	1.60s	2.53s	3.54s	4.74s	6.03s	7.58s	9.51s	10.9s
ω_2	0.23s	0.28s	0.69s	1.37s	2.16s	3.18s	4.16s	5.34s	6.87s	8.56s	10.1s
ω_3	0.26s	0.34s	1.08s	2.15s	3.68s	5.37s	7.47s	10.1s	13.7s	17.6s	20.9s
ω_4	0.46s	0.55s	1.33s	2.54s	3.95s	5.64s	7.70s	9.98s	12.7s	15.7s	18.4s
ω_5	0.72s	0.88s	1.90s	3.51s	5.60s	7.96s	10.8s	14.0s	18.1s	23.0s	27.0s

Table B.4.1: $\mathcal{W} \equiv$ *max*

References

[AA90] **Anantharaman, D.; Andrianarievelo, N.:** *Heuristical criteria in refutational theorem proving,* Proc. DISCO '90, LNCS 429, 1990, pp. 184-193.

[AD93] **Avenhaus, J.; Denzinger, J.:** *Distributing equational theorem proving,* Proc. 5th RTA, Montreal, CAN, LNCS 690, 1993, pp. 62–76

[ADF95] **Avenhaus, J.; Denzinger, J.; Fuchs, M.:** *DISCOUNT: A system for distributed equational deduction*, to appear in Proc. 6th RTA, Kaiserslautern, FRG, 1995

[BCP88] **Brock, B.; Cooper, S.; Pierce, W.:** *Analogical reasoning and proof discovery*, Proc. CADE 9, Argonne, IL, USA, 1988, LNCS 310, pp. 454–468

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

[Bu88] **Bundy, A.:** *The use of explicit plans to guide inductive proofs*, Proc. CADE 9, Argonne, IL, USA, 1988, LNCS 310, pp. 111–120

[Bu89] **Burstein, M.H.:** *Analogy vs. CBR: The purpose of mapping*, in: K.J. Hammond (ed.), Proceedings: Second case-based reasoning workshop (DARPA), Morgan Kaufmann, San Mateo, CA, USA, 1989, pp. 133–136

[Ca86] **Carbonell, J.:** *Derivational analogy: a theory of reconstructive problem solving and expertise acquisition*, in: R.S. Michalski et al. (eds.), Machine Intelligence; an AI approach, Vol. 2, 1986, pp. 371–392

[Ch93] **Christian, J.:** *Flatterms, discrimination nets, and fast term rewriting,* JAR 10, 1993, pp. 95–113

[Da88] **Davis, L. (ed):** *Genetic algorithms and simulated annealing*, Research notes in artificial intelligence, 1988

[De93] **Denzinger, J.:** *Teamwork: Eine Methode zum Entwurf verteilter, wissensbasierter Theorembeweiser*, Dissertation, FB Informatik, Universität Kaiserslautern, 1993

[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

[DS94a] **Denzinger, J.; Schulz, S.:** *Analysis and Representation of Equational Proofs Generated by a Distributed Completion Based Proof System*, SEKI-Report SR-94-05, University of Kaiserslautern, 1994

[DS94b] **Denzinger, J.; Schulz, S.:** *Recording, Analyzing and Presenting Distributed Deduction Processes*, Proc. PASCO '94, Linz, Austr., 1994

[Fu94] **Fuchs, M.:** *The application of goal-oriented heuristics for proving equational theorems via the unfailing Knuth-Bendix completion procedure. A case study: lattice ordered groups*, SEKI-Report SR-94-02, University of Kaiserslautern, 1994

[Ho75] **Holland, J.H.:** *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, Ann Arbor: Univ. of Michigan Press, 1975

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

[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

[KN93] **Koehler, J.; Nebel, B.:** *Plan modification versus plan generation*, Proc. IJCAI '93, Chambery, FRA, 1993, pp. 1436–1444

[KW94] **Kolbe, T.; Walther, C.:** *Reusing proofs*, Proc. 11th ECAI '94, Amsterdam, HOL, 1994, pp. 80–84

[Mc94] **McCune, W.W.:** *OTTER 3.0 reference manual and guide*, Techn. report ANL-94/6, Argonne Natl. Laboratory, 1994

[Ra91] **Rawlins, G. (ed):** *Foundations of genetic algorithms*, Morgan Kaufmann, 1991

[Sch66] **Schafer, R.D.:** *Introduction to non-associative algebra*, Academic Press, 1966

[SE90] **Suttner, C.; Ertel, W.:** *Automatic acquisition of search guiding heuristics*, Proc. CADE 10, Kaiserslautern, FRG, 1990, LNAI 449, pp. 470–484

[SF71] **Slagle, J.R.; Farrell, C.D.:** *Experiments in automatic learning for a multipurpose heuristic program*, Communications of the ACM, Vol. 14, Nr. 2, 1971, pp. 91–99

[Sm83] **Smith, S.F.:** *Flexible learning of problem solving heuristics through adaptive search*, Proc. IJCAI '83, Karlsruhe, FRG, 1983, pp. 422–425

[Su90] **Suttner, C.:** *Representing heuristic-relevant information for an automated theorem prover*, Proc. 6th IMYCS: aspects and prospects of theoretical computer science, LNAI 464, 1990

[Ta56] **Tarski, A.**: *Logic, Semantics, Metamathematics*, Oxford University Press, 1956

[Zh92] **Zhang, H.**: *Herky: High performance rewriting in RRL*, Proc. CADE 11, Saratoga Springs, NY, USA, 1992, LNAI 607, pp. 696–700

[Zh93] **Zhang, H.**: *Automated proofs of equality problems in Overbeek's competition*, JAR 11, 1993, pp. 333-351