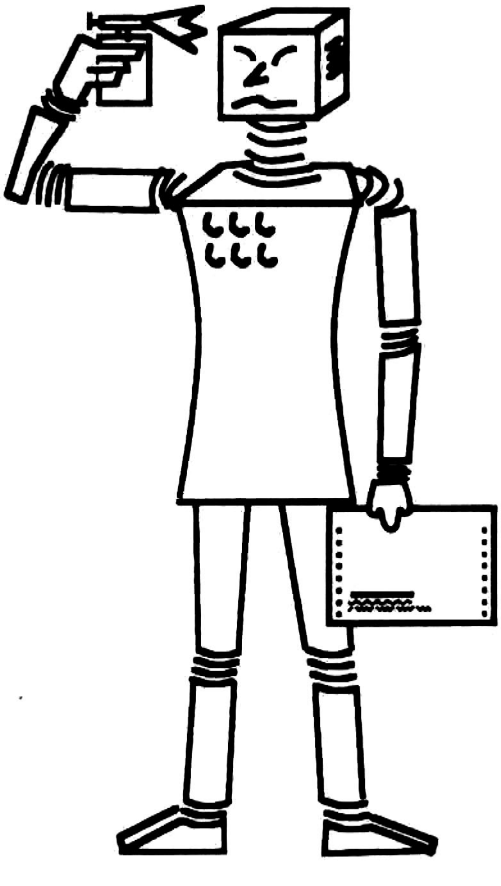
Fachbereich Informatik
Universität Kaiserslautern
D-67663 Kaiserslautern

SEKI - REPORT



High Performance ATP Systems by Combining Several AI Methods

Jörg Denzinger, Matthias Fuchs, Marc Fuchs

SEKI Report SR-96-09

High Performance ATP Systems by Combining Several AI Methods

Jörg Denzinger, Matthias Fuchs
Centre for Learning Systems and Applications (LSA)
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany
E-mail: {denzinge|fuchs}@informatik.uni-kl.de

Marc Fuchs
Fakultät für Informatik
TU München
80290 München
Germany
E-mail: fuchsm@informatik.tu-muenchen.de

October 29, 1996

Abstract

We present a concept for an automated theorem prover that employs a search control based on ideas from several areas of artificial intelligence (AI). The combination of case-based reasoning, several similarity concepts, a cooperation concept of distributed AI and reactive planning enables a system using our concept to learn form previous successful proof attempts. In a kind of bootstrapping process easy problems are used to solve more and more complicated ones.

We provide case studies from two domains of interest in pure equational theorem proving taken from the TPTP library. These case studies show that an instantiation of our architecture achieves a high grade of automation and outperforms state-of-the-art conventional theorem provers.

1 Introduction

Research concerned with achieving more efficient (fully automated) theorem provers focuses on three directions: higher inference rates, eliminating unnecessary inferences, and better control of the search. Each of these main directions has several subdirections. For example, one can achieve higher inference rates either by reducing the time spent for an inference (through more sophisticated implementation techniques) or by executing several inferences in parallel.

Although all these directions can indeed lead to more efficient theorem provers, better control of the search offers the highest gains in efficiency, but also causes the most problems and has some risks. Nearly all approaches to improving search control involve the use of techniques and methods from other areas of artificial intelligence (AI), as for example knowledge representation, case-based reasoning (CBR), learning, planning, or multi-agent systems. In most of the known works only ideas from one of these areas are exploited. For example, [Bu88] employs planning, [CMM90] multi-agent systems, and [KW94] explanation-based learning.

One area that should—from the human point of view—be the most promising for high efficiency gains is learning. The central point of the education of mathematicians is presenting important techniques by means of examples and learning these techniques by solving further related examples. But the use of machine-learning techniques for improving automated theorem provers faces several severe problems. First of all, learning is not enough. Learned knowledge has to be stored, the appropriate knowledge for solving a given problem has to be retrieved, and very often knowledge from different sources has to be combined. So, the focus of attention should not be restricted to the area of machine learning. Other areas of AI must also contribute in order to successfully apply the results the learning techniques produce.

In this paper we present an approach to controlling the search of an automated theorem prover that combines techniques form several areas of AI to overcome the problems that arise when trying to learn and to use control knowledge. The central idea is to utilize a known (i.e., learned) proof of a so-called *source problem* solved previously in order to guide the search for a proof of the *target problem* at hand. To this end we employ a method called *flexible re-enactment* (cp. [Fu96]).

Source problems must of course satisfy certain similarity criteria with respect to the target problem. Our techniques for maintaining a database of source problems and our mechanisms for selecting source problems that are the most similar to the target are inspired by CBR. Unfortunately, one of the important premises of CBR, namely that "*small differences between problems result in small differences of their solutions*", is not fulfilled in automated theorem proving. Therefore we have to expect that flexible re-enactment cannot always succeed.

We cope with this uncertainty by applying the TEAMWORK method ([AD93], [De95]), a multi-agent approach to distributed search. TEAMWORK reduces the risk of deploying an inappropriate heuristic by having a *team* of heuristics (agents) guide the search *concurrently* and *cooperatively*. The reactive planning capabilities of a further agent, namely the supervisor, are made use of to compose a suitable team (cf. [DK96]). More-over, the selection of the most suitable source problem required by flexible re-enactment can also be integrated with TEAMWORK in form of a specialized agent.

This specialized agent, called PES, employs a combination of similarity measures that allow for a good comparison (with respect to the target problem) even of problems that may share only some axioms with the target (after renaming symbols appropriately). As a result, PES can often suggest a source problem so that the flexible re-enactment of the source proof, together with some help of other agents, succeeds.

The combination of all these AI methods allowed us to build an automated theorem prover for pure equality reasoning that is fully automated, in both learning and proving, and is able to solve hard problems by using a kind of bootstrapping process that starts with easy problems and uses their proofs to gradually solve harder and harder ones. Besides providing the problems, no further interaction with the system is required. Our experiments validate and substantiate the achievements of our system.

2 Equational Reasoning

Equational reasoning deals with the following problem:

Given: A set $E = \{s_i = t_i \mid 1 \leq i \leq n \in \mathbb{N}\}$ of equations (of terms over a fixed signature *sig*) and a goal $u = v$.
Question: Is the goal equation a logical consequence of E, i.e., $E \models u = v$?

The completion method by Knuth and Bendix ([KB70], extended to unfailing completion, [BDP89], [HR87]) has proven to be quite successful for solving such a *proof problem* $\mathcal{A} = (E, u = v)$. The method is also a good example for so-called generating theorem provers, i.e., provers that are based on generating new facts until a fact describing the goal is reached. In the following, we assume the reader to be familiar with rewriting and completion techniques. For an overview see [AM90] or [DJ90].

As usual, a signature *sig* is a pair $(\mathcal{F}, \tau)$, where $\mathcal{F}$ is a set of operators and τ is a function $\tau : \mathcal{F} \rightarrow \mathbb{N}$ that returns the arity of an operator. The set of terms $T(\mathcal{F}, \mathcal{V})$ for a signature *sig* and a set $\mathcal{V}$ of variables is recursively defined by $x \in T(\mathcal{F}, \mathcal{V})$, if $x \in \mathcal{V}$, and $f(t_1, ..., t_n) \in T(\mathcal{F}, \mathcal{V})$ if $f \in \mathcal{F}$ with $\tau(f) = n$ and $t_1, ..., t_n \in T(\mathcal{F}, \mathcal{V})$.

The inference rules of a generating theorem prover can be divided into two classes: *extension* rules and *contraction* rules (see [De90]). Completion uses the expansion rule *critical-pair-generation* and the contraction rules *reduction* and *subsumption*. Basis for the completion procedure is a so-called reduction ordering $\succ$ that is used to restrict the applicability of the inference rules and to avoid cycles. In the following definitions, the right and left hand sides of equations may be exchanged.

A critical pair of two equations $l_1 = r_1$ and $l_2 = r_2$ is defined as the equation $\sigma(l_1)[p \leftarrow \sigma(r_2)] = \sigma(r_1)$, if p is a position in l_1 (and $l_1/p \notin \mathcal{V}$), so that σ is the mgu of l_1/p and l_2, and it holds true that $\sigma(l_1)[p \leftarrow \sigma(r_2)] \not\succeq \sigma(l_1)$ and $\sigma(r_1) \not\succeq \sigma(l_1)$. The resulting equation can be added to the set of equations, since it is a valid consequence of the two parent equations.

A reduction of an equation $s = t$ with an equation $l = r$ eliminates $s = t$ from the set of equations and replaces it with $s' = t$. A reduction may only be performed if there is a position p in s so that there is a match μ from l to s/p, i.e., $\mu(l) \equiv s/p$, and $\mu(l) \succ \mu(r)$. Then s' is given by $s' \equiv s[p \leftarrow \mu(r)]$. If there is no equation that can reduce a term of an equation, then term (and equation) are in *normal form*. The normalization of a term (or equation) is always a finite process.

The second contraction rule, subsumption, also concerns two equations $s = t$ and $l = r$ and totally eliminates $s = t$. The equation $l = r$ subsumes $s = t$—in symbols $l = r \lhd s = t$—if there is a position p in s so that there is a match μ so that $\mu(l) \equiv s/p$ and $t \equiv s[p \leftarrow \mu(r)]$.

An algorithmic realization of the inference rules of a generating theorem prover can be characterized as follows. There are two sets of facts (equations in our case): the set F^A of *active facts* and the set F^P of *passive facts* (the latter set may be represented implicitly). The algorithm centers on a main loop with the following body:

- select and remove a fact λ from F^P ("activate $\lambda \in F^P$")
- normalize λ (result is λ')
- if λ' is neither trivial nor subsumed then
  - normalize all elements of F^A
  - add λ' to F^A
  - add to F^P all facts that the inference rules can generate using λ' and elements of F^A

For equational reasoning, a proof is found, if the normalization of the two terms of the goal leads to the same term. If the set F^P becomes empty without this effect, then the goal is no logical consequence of the axioms.

If we assume that there is a given order in which contracting inference rules are applied, then the normalization of a fact is done deterministically. Hence, the remaining indeterminism is to determine which fact should be activated (i.e., selected from the set F^P of passive facts) next. In order to eliminate this indeterminism, selection strategies and heuristics are used (see for example [DF94]). In section 4 we present such a selection heuristic that is based on re-enacting a successful proof attempt for a problem that is somewhat *similar* to the problem at hand.

Since we want to learn from (successful) proof attempts in a domain of interest, we have to represent an actual proof run produced by our algorithm and a selection heuristic. In order to produce such a representation all steps of the prover have to be recorded (even all steps concerning elements of F^P, since such an element may be selected later in the proof run). This complete protocol $\mathcal{L}_\mathcal{A}$ of a proof problem $\mathcal{A}$ can become rather large. For learning purposes steps of the prover concerning elements of F^P that were not selected during the proof run are not important. If we eliminate these steps from $\mathcal{L}_\mathcal{A}$, which results in the *search protocol* $\mathcal{S}_\mathcal{A}$, the protocol becomes significantly smaller. But $\mathcal{S}_\mathcal{A}$ is not the actual proof of $\mathcal{A}$, since in $\mathcal{S}_\mathcal{A}$ there are also steps concerning selected facts that did not contribute to the proof (for most heuristics much more facts are selected than needed in the proof). If we eliminate these steps from $\mathcal{S}_\mathcal{A}$ we get a list $\mathcal{P}_\mathcal{A}$ that represents the proof actually found. The facts occurring in $\mathcal{P}_\mathcal{A}$ are also referred to as the set $P_\mathcal{A}$ of *positive facts* (in particular if we are not interested in their order of activation). The other facts in $\mathcal{S}_\mathcal{A}$ are the negative facts N that are needed for some learning approaches (see [Fu95a], [Fu96]). In order to be able to reproduce a run we store a successful proof attempt as quadruple $(\mathcal{A},\mathcal{H},\mathcal{S}_\mathcal{A},\mathcal{P}_\mathcal{A})$, where $\mathcal{H}$ is the heuristic used. So, the use of various approaches for learning from previous proof experience is permitted.

3 Teamwork

The TEAMWORK method is a knowledge-based distribution method for certain search processes ([De95]). Equational deduction by completion, as well as for example first-order deduction by (hyper-) resolution, is a member of this class of search processes.

In a TEAMWORK-based system there are four different types of agents: experts, specialists, referees, and a supervisor. Experts and specialists are the agents that work on really solving a given problem. *Experts* form the core of a team. They are problem solvers (in our case theorem provers) that use the same inference mechanism (in our case unfailing completion), but different selection strategies for the next inference step to do. Therefore each expert can work with the search state of each other expert (but will obviously search in its own direction). In our case, a search state is represented by the sets F^A and F^P of active and passive facts.

Specialists can either also search for a solution of the given problem or a part of it, or they can perform administrative tasks that require time-consuming computations that help the supervisor do its work. They may even combine these two tasks. In case of a theorem prover, specialists may use other inference mechanisms than the experts to generate new facts, or they may analyze and classify the given problem like our specialist PES does (see section 5.2).

Each expert and specialist of a team needs its own computing node and works independent of the others for a given time, the so-called *working period*. Typically, in a TEAMWORK-based system there are much more experts and specialists than available computing nodes. Therefore, the supervisor determines the subset of experts and specialists that are active during a working period.

After a working period a *team meeting* takes place. In the first phase of a team meeting, the *judgment phase*, each active expert and specialist is evaluated by a referee. Each referee has two tasks: judging the whole work of the expert/specialist of the last working period and selecting outstanding results. The first task results in a *measure of success*, an objective measure that allows the supervisor to compare the experts. The second task is responsible for the cooperation of the experts and specialists, since each selected result will be part of the common start search state of the next working period. For both tasks statistical criteria (with small extensions) have proven to be sufficient (see [DF96]). The referees send the results of their work, together with special information generated by some specialists, to the supervisor.

After the *supervisor* has received the results of the referees, the *cooperation phase* of a team meeting begins. During this phase, the supervisor has to construct a new starting point for the next working period, select the members of the team for this next period and determine the length of the period. The new start state for the whole team is constructed by taking the search state $S = (F^A, F^P)$ of the expert with the best measure of success and by enriching this state with the selected results of the other experts and the specialists. In case of a theorem prover results are facts, and the enrichment is achieved by adding selected results to F^P and immediately activating them.

The supervisor determines the members of the next team with a reactive planning process involving general information about experts, specialists, referees and domains of interest (the *long-term memory*) and actual information on the performance of (former or current) team members regarding the given problem (i.e., the measures of success which form the *short-term memory*). Which general information to use is decided with the help of the results of some specialists that determine which parts of the general information affect the actual problem. The long-term memory often suggests a *plan skeleton* that contains several small teams for different phases of a proof attempt. These suggested teams are reinforced with appropriate experts/specialists (if more computing nodes are available). If such a plan is put to work, then during each team meeting the plan has to be updated. This means that adjustments are made according to the actual results. These adjustments range from exchanging only one or a few experts over moving on to the next phase of the plan skeleton to totally changing the plan (see [DK96]).

The length of the next working period is also determined by the reactive planning process. Depending on the extent of the changes in the team, its success and the size of the new start state, a basic time is either shortened or lengthened. When the supervisor has fulfilled these three tasks it sends the new start state to all computing nodes (after experts/specialists and referees were assigned to them following the previous decision) and the team meeting ends.

TEAMWORK allows (even without experts and specialists using learned knowledge) for synergetic effects that result in enormous speed-ups and in finding solutions to problems that are beyond the possibilities of the single experts and specialists. While the competition of the experts directs the whole team into interesting (and promising) parts of the search space, the cooperation provides the experts with excellent facts they are not able to come up with alone. Thus gaps in their derivations towards the goal can be closed.

Obviously, TEAMWORK solves many of the problems that the use of learned knowledge causes. Applicable learned knowledge can be detected by specialists. Applicable knowledge that does not lead towards solving the actual problem is detected and not used anymore. Instead, the whole system is adapted to the given problem by the reactive planning process. Gaps in derivations that are caused by using similarity with respect to only a certain aspect between a source problem and the given problem can be filled through cooperation with other experts. In general, even learning is easier, because it is only necessary to learn single aspects or parts of the solution of a source problem. These parts can be put together by forming a team.

As already stated, learning requires representations of found proofs and the runs that led to these proofs. Distributed provers cause additional problems that are solved in TEAMWORK-based systems by means of so-called *reproduction runs* (see [DS96]). In order to make such reproduction runs possible, during each (conventional) run a small protocol $\mathcal{C}$ is written that contains all necessary information (i.e., which expert has made how many steps in which working periods, which referees were used, etc.) to reproduce this run. So $\mathcal{C}$ can be seen as the description of a "hyper-heuristic" and is used instead of a single heuristic $\mathcal{H}$ when storing data regarding distributed proofs.

4 Flexible Re-enactment

Similarity between two proof problems $\mathcal{A}$ and $\mathcal{B}$ can occur in many variations. One possible kind of similarity is that a considerable number of the facts that contribute to a proof of $\mathcal{A}$ are also useful for proving $\mathcal{B}$ (or vice versa). This means in our terminology that the associated sets of positive facts $P_{\mathcal{A}}$ and $P_{\mathcal{B}}$ or the proofs $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_{\mathcal{B}}$ "have a lot in common" or, in other words, share many facts. ("$P_{\mathcal{A}} \cap P_{\mathcal{B}}$ is almost equal to $P_{\mathcal{A}}$ and $P_{\mathcal{B}}$.") Our goal is to think up a heuristic that is able to exploit such a similarity.

Given $\mathcal{I} = (\mathcal{A}_S, \mathcal{H}, \mathcal{S}_{\mathcal{A}_S}, \mathcal{P}_{\mathcal{A}_S})$ as past experience regarding a source problem $\mathcal{A}_S$, and assuming that a target problem $\mathcal{A}_T$ is similar to $\mathcal{A}_S$ in the way just described, it is reasonable to concentrate on mainly deducing facts when attempting to prove $\mathcal{A}_T$ that also played a role in finding the source proof $\mathcal{P}_{\mathcal{A}_S}$, namely the positive facts $P_{\mathcal{A}_S}$. We therefore design a heuristic FlexRE which—when trying to prove $\mathcal{A}_T$—makes use of $\mathcal{I}$ by giving preference to facts that were important for finding $\mathcal{P}_{\mathcal{A}_S}$. Such facts will henceforth be referred to as *focus facts*. Note that focus facts are facts inferred or inferable in connection with $\mathcal{A}_T$. They must be distinguished from the positive facts $P_{\mathcal{A}_S}$ belonging to the source problem $\mathcal{A}_S$, since it might be the case that some $\lambda \in P_{\mathcal{A}_S}$ is not deducible at all in connection with $\mathcal{A}_T$ (due to a different axiomatization). $P_{\mathcal{A}_S}$ is merely used to determine if some fact λ inferable in connection with $\mathcal{A}_T$ is a focus fact. To put it another way, the use of $P_{\mathcal{A}_S}$ is effected by FlexRE on a strictly heuristic basis, meaning that $P_{\mathcal{A}_S}$ only influences the selection of facts from F^P, not, for instance, F^P itself. That is, $P_{\mathcal{A}_S}$ is a guideline that FlexRE tries to follow if possible.

Nevertheless, our following definitions (paired with sensible parameter settings) will guarantee that, if the proof of $\mathcal{A}_S$ is also a proof of $\mathcal{A}_T$, then no (or at most a negligible amount of) unnecessary facts will be selected. In contrast to [KW94], a "search" still takes place, but the computational effort is minimal (due to the then (nearly) perfect control). Naturally, the strength of FlexRE consists in being able to deal with cases in which another, but quite similar proof is needed (which is beyond the scope of [KW94]).

Depending on how strongly focus facts are preferred, FlexRE will re-enact (parts of) $\mathcal{P}_{\mathcal{A}_S}$ more or less quickly. Some of the focus facts, though useful for proving $\mathcal{A}_S$, may be irrelevant regarding the proof $\mathcal{P}_{\mathcal{A}_T}$ of $\mathcal{A}_T$ eventually found. But these irrelevant focus facts are not a big problem. The crucial difficulty is to find those (non-focus) facts that have to supplement the relevant focus facts in order to obtain a proof $\mathcal{P}_{\mathcal{A}_T}$. It is very likely that these (few) missing facts are descendants of relevant focus facts.

Consequently, FlexRE should also favor descendants of focus facts. Favoring descendants should weaken with their "distance" from focus facts, since it cannot be assumed that the few missing facts are located very deeply relative to focus facts.[1]

Preferring descendants of focus facts in addition to giving preference to focus facts themselves justifies the attribute 'flexible' in the term 'flexible re-enactment' which summarizes the working method of FlexRE. We shall now explain details.

For the definition of FlexRE the notions 'difference' and 'distance' are pivotal. First, we define the *difference diff* between two facts λ and λ' in order to make the notion 'focus fact' computationally tangible:

$$\mathit{diff}(\lambda,\lambda') = \begin{cases} 0, & \lambda \lhd \lambda' \\ 100, & \text{otherwise.} \end{cases}$$

For the time being, we content ourselves with this simple definition of difference. Note that the values 0 and 100 are somewhat arbitrary but intuitive hints of percentages, denoting 'no difference' and 'total difference', respectively. Or, expressed with opposite terms, 0 and 100 represent "perfect similarity" and "no similarity at all", respectively. As we shall see, the restriction of *diff* to $\mathbb{N}_{100} = \{0, 1, \ldots, 100\}$ entails that all further computations will produce values from $\mathbb{N}_{100}$, which makes computations more transparent and easier to interpret and to handle than, for instance, computations involving unbounded values.

diff is used to find out whether a given fact λ is a focus fact. We define

$$\mathcal{D}(\lambda) = \min\Big(\{\mathit{diff}(\lambda,\lambda') \mid \lambda' \in P_{\mathcal{A}_S}\}\Big)\,.$$

Hence, $\mathcal{D}(\lambda)$ returns the minimal difference between a given fact λ (target) and the positive facts (source). If $\mathcal{D}(\lambda) = 0$ then λ is considered to be a focus fact, which complies with the ideas from above.

Now recall that also descendants of focus facts are to be favored. The preference given to them should, however, decrease with their distance from focus facts. The *distance* $d(\lambda)$ of a given fact λ measures distance, roughly said, in terms of the number of inference steps separating λ from ancestors which are focus facts. It depends on the distance of the immediate ancestors of λ from focus facts (if λ is not an axiom) and $\mathcal{D}(\lambda)$:

$$d(\lambda) = \begin{cases} \psi\Big(q, \mathcal{D}(\lambda)\Big), & \text{if } \lambda \text{ is an axiom} \\ \psi\Big(d(\lambda_1), \mathcal{D}(\lambda)\Big), & \text{if } \lambda_1 \text{ is the (only) immediate ancestor of } \lambda \\ \psi\Big(\gamma(d(\lambda_1), d(\lambda_2)), \mathcal{D}(\lambda)\Big), & \text{if } \lambda_1 \text{ and } \lambda_2 \text{ are the immediate ancestors of } \lambda. \end{cases}$$

The first argument of ψ represents the distance of the immediate ancestors, which is simply given as the distance of the immediate ancestor if there is just one immediate ancestor. If λ is an axiom (i.e., there are no ancestors), this value is specified by a parameter $q \in \mathbb{N}_{100}$. If λ has two immediate ancestors, then γ computes this value.

[1] "Distance" and (relative) depth basically refer to the number of inference steps separating two facts, one of these facts contributing to the deduction of the other.

We chose a parameterized γ employing a parameter $q_1 \in [0;1]$. Depending on q_1, the result of γ ranges between the minimum, the average, and the maximum of the distances of the immediate ancestors:

$$\gamma(x,y) = \min(x,y) + \lfloor q_1 \cdot (\max(x,y) - \min(x,y)) \rfloor \ .$$

Using $q_1 = 0$ or $q_1 = 1$, γ computes the minimum or maximum, respectively. With $q_1 = 0.5$, γ computes the (integer part of the)[2] average.

The distance of immediate ancestors (or q) and $\mathcal{D}$ are combined by ψ yielding $d(\lambda)$. ψ should—for obvious reasons—satisfy the following criteria: On the one hand, $d(\lambda)$ should be minimal (i.e., 0) if $\mathcal{D}(\lambda) = 0$, in which case λ itself is a focus fact. On the other hand, the value produced by ψ should increase (reasonably) with the values obtained from γ and $\mathcal{D}$ in order to reflect the (growing) remoteness of λ from focus facts (and, in a way, from the source proof). As a matter of fact, γ already satisfies the latter criterion. Therefore, ψ is in parts identical to γ. It also uses a parameter $q_2 \in [0;1]$.

$$\psi(x,y) = \begin{cases} 0, & y = 0 \\ \min(x,y) + \lfloor q_2 \cdot (\max(x,y) - \min(x,y)) \rfloor, & \text{otherwise.} \end{cases}$$

The remaining task consists in designing FlexRE so that it offers a reasonable degree of specialization in (i.e., focus on) the source proof that is paired with an acceptable degree of flexibility, i.e., the ability to cope with a target problem with a proof that requires minor to moderate deviations from the source proof. The use of d already provides sufficient specialization by directing the search towards the source proof. Its rudimentary flexibility can be enhanced by combining it with some (basic) heuristic $\mathcal{H}$ giving FlexRE.

Among several sensible alternatives we picked the following:

$$\boxed{\mathsf{FlexRE}(\lambda) = (d(\lambda) + p) \cdot \mathcal{H}(\lambda)\,, \quad p \in \mathbb{N}}$$

The parameter p controls the effect of $d(\lambda)$ on the final weight FlexRE(λ). $d(\lambda)$ will be dominant if $p = 0$. In this case, if $d(\lambda) = 0$, FlexRE(λ) will also be 0 regardless of $\mathcal{H}(\lambda)$. As p grows, $\mathcal{H}$ increasingly influences the final weight, thus mitigating the inflexibility of the underlying method, namely using $d(\lambda)$ *alone* as a measure of the suitability of a fact λ. For very large p, the influence of $d(\lambda)$ becomes negligible, and FlexRE basically degenerates into $\mathcal{H}$.

Finally, we would like to emphasize that flexible re-enactment can achieve much more than can be achieved by simply adding (positive) facts of a source proof as lemmas to the target axiomatization. Adding lemmas is allowed only if the equivalence of target and source axiomatization is known. Flexible re-enactment does not depend on this

[2] We restrict our computations to $\mathbb{N}$, because there is no gain in "high precision arithmetic" when dealing with weighting functions, but there would be a loss in efficiency w.r.t. computation time.

requirement. As a matter of fact, flexible re-enactment also proves very useful if target and source axioms agree merely partially (cp. [Fu96]). But even when source and target axioms are the same, experiments documented in [Fu95b] reveal that flexible re-enactment is superior to adding lemmas. The superiority mainly originates from favoring focus facts (lemmas) and their descendants, which is an important property of flexible re-enactment that keeps the search close to the source proof. This focused search cannot be achieved by simply adding lemmas. Naturally, if the source is inappropriate, focusing on the source proof will be counterproductive.

5 Learning and CBR in the Teamwork Environment

In sections 3 and 4 we concentrated on how to use knowledge learned from previous successful proofs (in form of flexible re-enactment) and on how to overcome the problems such a use might cause (in form of the TEAMWORK method with cooperation with other experts, assessment of experts and results, and reactive planning to adapt to the problem at hand using long- and short-term memory). The problems that remain are how to find a proof that should be (flexibly) re-enacted in order to solve a given target problem and how to structure, build, and maintain the long-term memory from proof run to proof run.

The first problem will be tackled by a specialist PES that is providing the supervisor with information about known proof problems that are similar to the given target problem (see subsection 5.2). The second problem naturally depends on how the proof problems are presented to the system. Found proofs have simply to be extracted, analyzed and stored (the latter depending on how specialist PES will perform its retrieval). As we shall see in the next subsection, the necessary components are already provided in form of TEAMWORK agents.

5.1 The Basic Learning Cycle

Systems that use learning techniques for solving their tasks can be (very) roughly divided into two groups: systems that have a clearly defined learning phase after which (usually) no further learning takes place, and systems that always learn. In automated theorem proving, systems of the first type may be usable in clearly defined situations (see, for example, [Fu95a]), but in general, as in the case of mathematicians, learning should never stop.

Nevertheless, one can observe times in the use of a (learning) theorem prover in which new domains of interest are explored, and other times in which one is interested in proving one particular problem (perhaps without any a priori knowledge which domain it belongs to). When exploring a new domain, typically there is a set of example problems to be solved, and when starting the exploration no knowledge in the prover will be triggered. In the following, we will first concentrate on the exploration of a new domain and then we will point out how the one-problem case is handled.

When exploring a new domain the ordering of the problems given to a prover may influence its success. Even when teaching (human) students of mathematics there are very often several different ways to introduce a concept, and it depends on the respective student which ways result in successful learning. In the case of a learning automated theorem prover this problem is particularly grave, since the calculus used by the prover very seldom is also the calculus a human teacher uses. This can have the effect that problems a human being thinks of as easy are very difficult for the prover and vice versa.

In order to deal with this problem we decided to let the prover handle the ordering in which it tries to solve a given set of problems and also allow the prover to make several attempts to solve a problem. The latter is necessary since each solved problem may result in new knowledge that allows for solving some other problems that could not be solved so far ("*bootstrapping*"). Note that the set of problems given to the prover has to include easy and typical problems of a domain of interest that the prover can use to get fundamental knowledge about the domain. This is similar to a human student who cannot be expected to work on difficult problems of a domain without having learned the basics of it before.

As already stated, in a TEAMWORK-based system the long-term memory that represents knowledge about domains of interest is the responsibility of the supervisor. When confronted with a set of example problems of a new domain, the supervisor controls not only the single proof attempts, but a whole series of proof attempts that are to result in solving as many of the problems as possible.

Since the supervisor has no appropriate information when being confronted with a new domain, the first step is to try to solve the given problems with conventional means, i.e., without the use of experts and specialists that employ learned knowledge. This is accomplished by using a pre-defined team for a few cycles for each of the given problems (in a separate run). Those problems that could be solved by this team in the given period of time will be rerun in the so-called *reproduction mode* using the small protocol $\mathcal{C}$. Note that generating a complete protocol is rather expensive (both in run time and disk space needed), so that this mode is only used if one is really interested in a proof (that was already found by the prover). See [DS96] for details on this topic.

After generating $\mathcal{I} = (\mathcal{A}, \mathcal{C}, \mathcal{S}_\mathcal{A}, \mathcal{P}_\mathcal{A})$ for each solved problem $\mathcal{A}$, this data is integrated into a database of past proof experience that is part of the long-term memory of the system. This database is essentially organized as case base. Hence, the structure and retrieval processes regarding this database are strongly related to CBR techniques (cp. [Ko92] and subsection 5.2). Then the supervisor tries to solve the remaining problems (again imposing a time limit on each run), but now the teams are different. The team of the first working period is again pre-defined, and contains, besides good general purpose experts, the specialist PES.

After the first working period, the supervisor uses its reactive planning process to adapt the team to the problem. If PES was not able to report to the supervisor a problem from the case base that is similar to the target problem at hand, then the supervisor proceeds according to its standard procedure (i.e., trying to adapt the team to the problem at hand, cp. [DK96]). Otherwise, the expert FlexRE will become a member of

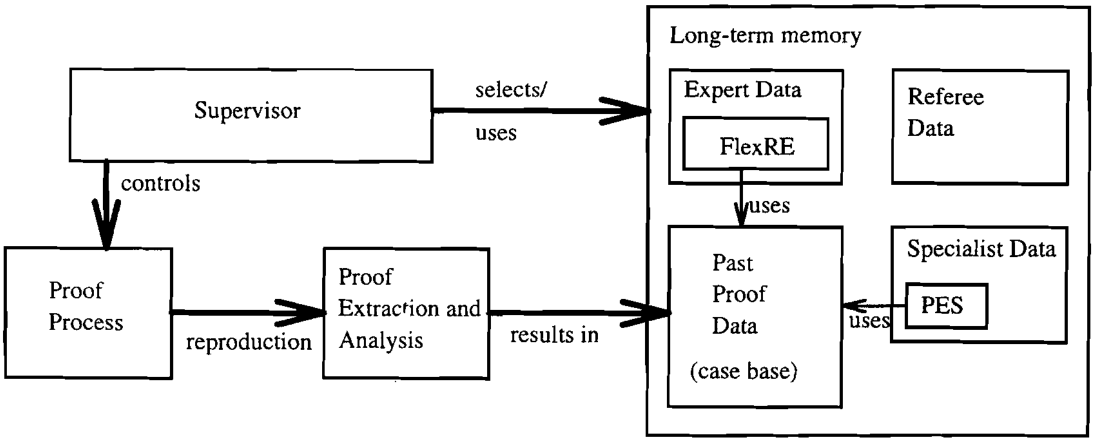


Figure 1: The learning cycle

the next team, utilizing the reported problem from the case base as source problem.[3] Problems that can now be solved (due to the use of previous proof experience and the adaptation to the given problem) are rerun again to produce new data for the case base. For the remaining unsolved problems this whole process is repeated until no more new problems can be proven.

Note that after the initial round, that uses the pre-defined team without components using learned knowledge, in all other rounds each solved problem is immediately added to the case base so that it already can be used for the proof attempt of the next problem. This way, the number of rounds is often reduced.

In the one-problem case, i.e., in the presence of fundamental domain knowledge, the supervisor immediately employs a team that includes PES, and the supervisor plans and controls the whole proof attempt as described above. A time limit for the attempt can be provided by the user. If the system is successful, the data of the run is also added to the case base. The whole learning cycle (with the structure of the long-term memory and the components that use the case base) is depicted by figure 1.

5.2 Specialist PES

As described before, it is the main task of specialist PES to retrieve one or possibly several source problems that are similar to a given target problem $\mathcal{A}_T$, and to transmit information on these problems to the supervisor. More exactly, after receiving a target problem $\mathcal{A}_T = (Ax_T, \lambda_T)$ given over a fixed signature sig_T, PES returns information $R_{\text{PES}}(\mathcal{A}_T)$ on similar problems $\mathcal{A}_S$, where $R_{\text{PES}}(\mathcal{A}_T) = \{(\sigma, P_S) : cond_T(\sigma(\mathcal{A}_S))\}$. The data is determined by $cond_T$ and comprises the set of positive facts P_S associated with source problem $\mathcal{A}_S$, and a signature match σ from $\mathcal{A}_S$ to $\mathcal{A}_T$. $cond_T$ denotes

[3] If PES provides more than one possible source problem, then the supervisor can react in various ways: It can select the most similar source problem (breaking ties arbitrarily) and discarding all others—which is what we did in our experiments—, or it may use several experts FlexRE, each using a different source problem (without exceeding the number of available computing nodes of course), or it may supply the single FlexRE with the source problems provided by PES in succession, limiting each attempt to a reasonable number of working phases.

that a problem $\mathcal{A}_S$ (translated from sig_S to sig_T by applying σ) is most similar to $\mathcal{A}_T$ (there are no other problems more similar to $\mathcal{A}_T$ than $\mathcal{A}_S$), but also that the similarity between $\mathcal{A}_S$ and $\mathcal{A}_T$ seems to be sufficient (details are given below).

σ is an injective, arity-preserving function that maps the function symbols of signature sig_S to the function symbols of signature sig_T. Note that σ can be easily extended to terms, equations, sets of equations, and finally proof problems. Usually there exists more than one signature match. σ makes us independent of possible renaming of function symbols. This is necessary to employ syntactic criteria to measure similarity, which are in particular applied by flexible re-enactment (cp. section 4), but also by $cond_T$ to be discussed in the sequel.[4]

Recall that the predicate $cond_T$ determines the source problems (in the case base) that should be transmitted to the supervisor. We are interested in a predicate $cond_T$ defined for proof problems over sig_T with $cond_T(\sigma(\mathcal{A}_S))$ if (and only if) there exists no other source problem more similar to $\mathcal{A}_T$ than $\mathcal{A}_S$, and the similarity between $\mathcal{A}_T$ and $\mathcal{A}_S$ is sufficient (i.e., exceeds some threshold), so that it is very likely that flexible re-enactment will be successful. In order to construct such a predicate we shall use a quasi-ordering $\succeq_T$ that allows us to compare the similarity between proof problems and a target problem. It should hold true that $\mathcal{A}_1 \succ_T \mathcal{A}_2$ if (and only if) $\mathcal{A}_T$ is more similar to $\mathcal{A}_1$ than to $\mathcal{A}_2$. Furthermore, an absolute measure of similarity is needed in order to construct a *minimal similarity* predicate ms that estimates whether the similarity between the target and a source problem seems to be sufficient or not.

Basically, there are two possibilities to estimate whether a problem $\mathcal{A}_1$ can provide a more suitable source proof than a problem $\mathcal{A}_2$. These two possibilities that can be used to realize $\succeq_T$ are a comparison of problem descriptions and a comparison of the search effort spent on solving the two problems. Considering the working method of flexible re-enactment, it is reasonable to consider $\mathcal{A}_1$ as providing a more suitable source proof than $\mathcal{A}_2$, if the target $\mathcal{A}_T$ is more similar to problem $\mathcal{A}_1$ than to $\mathcal{A}_2$ with respect to its problem description. Nevertheless some information on the search conducted to solve a problem, namely the length of the search protocol $\mathcal{S}$, can be consulted. Experiments have shown that often more difficult proof problems (problems with a longer search protocol) are more useful for flexible re-enactment than easier ones. This is understandable, because a longer search protocol usually indicates that some of the positive facts were quite difficult to reach when proving a problem. Considering this, it seems to be sensible to force the activation of these probably useful, but hard to reach facts by using them as focus facts (cp. section 4).

Similarity between problem descriptions is surely also suitable for constructing an absolute similarity value in order to decide in favor of or against sufficient similarity. But information on the search protocol is difficult to use for this particular purpose. Thus the length of the search protocol can be useful to compare different problems, but using it as a criterion for deciding whether the similarity to a target is sufficient or not does not appear to be sensible.

[4] If there is no signature match from sig_S to sig_T, then problem $\mathcal{A}_S$ cannot be used as a source problem for target $\mathcal{A}_T$.

In order to assess the differences between the problem descriptions of two proof problems we employ the similarity measure sim_T. As discussed before, this measure is useful to construct $\succeq_T$, but also to measure the degree of similarity between a problem and the target so as to obtain the predicate ms. The design of sim_T is motivated by the fact that a target problem $\mathcal{A}_T = (Ax_T, \lambda_T)$ is proved by virtue of a proof of a source problem $\mathcal{A}_S = (Ax_S, \lambda_S)$, if all axioms of Ax_S are subsumed by axioms of the target problem, and the target goal λ_T is subsumed by the source goal λ_S. Hence, subsumption criteria like this will play the major role in our approach. Furthermore, we refine our measure sim_T by using several other criteria like subsumption modulo some theory (e.g., AC) or homeomorphic embedding. These refinements have proven their usefulness in experiments, although naturally a simple proof replay often is impossible if Ax_S and Ax_T are similar in such a way.

In order to realize measure sim_T we introduce a (asymmetric) similarity rating sim_T^{eq} defined on equations over sig_T.

Definition 5.1 (sim_T^{eq}) *Let ax_1 and ax_2 be two equations in $Term(\mathcal{F},\mathcal{V})^2$. Let further A denote some theory (e.g., AC). We define $sim_T^{eq} : Term(\mathcal{F},\mathcal{V})^2 \times Term(\mathcal{F},\mathcal{V})^2 \to [0;1]$ by*

$$sim_T^{eq}(ax_1, ax_2) = \begin{cases} 1 & ; ax_1 \lhd ax_2 \\ 0.8 & ; ax_1 \lhd_A ax_2 \wedge ax_1 \ntriangleleft ax_2 \\ 0.2 & ; (ax_1 \lhd_{HOM} ax_2 \vee ax_2 \lhd_{HOM} ax_1) \wedge ax_1 \ntriangleleft ax_2 \wedge ax_1 \ntriangleleft_A ax_2 \\ 0 & ; \textit{otherwise.} \end{cases}$$

$\lhd$ denotes "plain" subsumption, and $\lhd_A$ subsumption modulo the theory given by A. (Here, we shall always use $A = AC$. Note that for testing $\lhd_{AC}$ we basically assume every binary function symbol f to be AC, although f need not actually be AC. Hence the lesser "reliability" value 0.8. Subsumption modulo AC nevertheless makes sense since often certain sub-structures are AC.) $ax_1 \lhd_{HOM} ax_2$ stands for a homeomorphic embedding of ax_2 in ax_1. Refinements of sim_T^{eq} are possible by adding further similarity criteria. (Refinements are actually recommendable in order to increase the ability of sim_T^{eq} to produce distinctive measures. A description of these technical details is beyond the scope of this report.) The values of sim_T^{eq} represent the importance of the similarity criteria we use. As described above subsumption is considered to be very important, whereas homeomorphic embedding is considered to be a very weak similarity criterion. Note that $sim_T^{eq}(ax_1, ax_2)$ does not depend on ax_1 and ax_2 being equations, but can basically be applied to any kind of formula.

With the help of this measure we are able to construct a similarity measure defined on proof problems:

Definition 5.2 (sim_T) *Let $\mathcal{A}_T = (Ax_T, \lambda_T)$ be a target problem, and $\mathcal{A}_S = (Ax_S, \lambda_S)$ be a source problem given over sig_T. Let $Ax_T = \{ax_1, \ldots, ax_m\}$, $Ax_S = \{ax'_1, \ldots, ax'_n\}$ $(n, m > 0)$. The similarity of target and source problem is $sim_T(\mathcal{A}_T, \mathcal{A}_S) = (s_1, s_2, s_3) \in [0;1]^3$, where*

$$s_1 = \frac{1}{n} \cdot \sum_{i=1}^{n} \max\{sim_T^{eq}(ax, ax'_i) : ax \in Ax_T\}$$

$$\begin{aligned} s_2 &= \frac{1}{m} \cdot \sum_{i=1}^{m} \max\{sim_T^{eq}(ax_i, ax) : ax \in Ax_S\} \\ s_3 &= sim_T^{eq}(\lambda_S, \lambda_T) \end{aligned}$$

Thus, s_1 judges the degree of "coverage" of Ax_S through similar axioms of Ax_T. For example, we have $s_1 = 1$ if all source axioms are subsumed by target axioms. Naturally, s_1 decreases if only weaker similarity criteria are fulfilled. The value s_2 represents the percentage of target axioms that have a similar counterpart in Ax_S. Additional axioms in Ax_T do not prevent the source proof from being applicable (in the case $s_1 = 1$), but may complicate the search for this proof. Finally, s_3 measures the similarity between target and source goal λ_T and λ_S. For example, we have $s_3 = 1$ if λ_S subsumes λ_T.

With the help of sim_T we are now able to estimate if the similarity between $\mathcal{A}_T$ and $\mathcal{A}_S$ seems to be sufficient. To this end, we check if the predicate ms is fulfilled with $ms(sim_T(\mathcal{A}_T, \mathcal{A}_S))$ iff $c_1 \cdot s_1 + c_2 \cdot s_2 + c_3 \cdot s_3 \geq min$, where $c_1, c_2, c_3 \in \mathbb{R}$, and $min \in \mathbb{R}$ is the threshold. In our implementation we use $c_1 = 3$, $c_2 = 1$, $c_3 = 2$, and $min = 1$. Hence additional axioms in Ax_T are considered quite harmless, while a good coverage of Ax_S is considered to be important. Since we use $min = 1$, a subsumption of one third of the axioms of Ax_S, or no superfluous target axioms, or a subsumed target goal each suffice alone to reach the threshold.

sim_T is also employed to define $\succeq_T$. We define $\succeq_T$ as a lexicographic combination of two quasi-orderings $\geq_S$ and $\geq_D$, where $\geq_S$ compares the similarity of two problem descriptions with respect to the target problem, and $\geq_D$ compares the length of the search protocols.

Definition 5.3 ($\succeq_T$) *Let $\mathcal{A}_T$ be a target problem, and $\mathcal{A}_1$ and $\mathcal{A}_2$ be two source problems over sig_T. $\mathcal{S}_i$ denotes the search protocol obtained when solving $\mathcal{A}_i$ ($i \in \{1,2\}$). Let $sim_T(\mathcal{A}_T, \mathcal{A}_1) = (s_1, s_2, s_3)$, $sim_T(\mathcal{A}_T, \mathcal{A}_2) = (s_1', s_2', s_3')$. We define $\geq_S$ by*

$$\mathcal{A}_1 \geq_S \mathcal{A}_2 \quad \textit{iff} \quad (s_1, s_2, s_3) = (s_1', s_2', s_3') \lor (s_1 > s_1' \land s_2 > s_2' \land s_3 > s_3').$$

Furthermore, we define $\geq_D$ by

$$\mathcal{A}_1 \geq_D \mathcal{A}_2 \quad \textit{iff} \quad |\mathcal{S}_1| \geq |\mathcal{S}_2|.$$

Finally, $\succeq_T$ is defined as the lexicographic combination of $\geq_S$ and $\geq_D$.

Using $\succeq_T$ and ms, we can define $cond_T$.

Definition 5.4 ($cond_T$) *Let $\mathcal{A}_T$ and $\mathcal{A}_S$ be a target and source problem given over sig_T. $cond_T$ is defined as follows:*

$$cond_T(\mathcal{A}_S) \quad \textit{iff} \quad ms(sim_T(\mathcal{A}_T, \mathcal{A}_S)) \land \neg\exists\mathcal{A} : \mathcal{A} \succ_T \mathcal{A}_S.$$

In the subsequent section we shall discuss the experimental results we obtained with DISCOUNT by using specialist PES and expert FlexRE. But at first we want to illustrate the previous definitions with the following example.

Example 5.1 *Let sig* $= (\mathcal{F}, \tau)$ *be a signature with* $\mathcal{F} = \{+, -, 0\}$ *and* $\tau(+) = 2$, $\tau(-) = 1$, *and* $\tau(0) = 0$. *Let* $\mathcal{A}_{S_1} = (Ax_{S_1}, \lambda_{S_1})$ *and* $\mathcal{A}_{S_2} = (Ax_{S_2}, \lambda_{S_2})$ *be two solved source problems given over sig. Furthermore, let* $\mathcal{A}_T = (Ax_T, \lambda_T)$ *be a target problem over sig. The proof problems are defined as follows:*

$$\begin{array}{lrcl@{\qquad}lrcl}
Ax_{S_1}: & x+(y+z) & = & (x+y)+z & Ax_{S_2}: & x+(y+z) & = & (x+y)+z \\
 & x+0 & = & x & & 0+x & = & x \\
 & x+(-x) & = & 0 & & (-x)+x & = & 0 \\
\lambda_{S_1}: & -(x+y)+0 & = & (-y)+(-x) & \lambda_{S_2}: & -(x+y) & = & (-y)+(-x)
\end{array}$$

Furthermore, $Ax_T = Ax_{S_2}$ *and* $\lambda_T \equiv -(x+0) = (-0)+(-x)$. *Rating the similarity between the source problems and the target results in*

$$\begin{array}{ll}
sim_T(\mathcal{A}_{S_1}, \mathcal{A}_T) = (s_1, s_2, s_3) \textit{ with} & s_1 = \frac{1}{3}\cdot(1+0.8+0.8) = 0.86 \\
 & s_2 = \frac{1}{3}\cdot(1+0.8+0.8) = 0.86 \\
 & s_3 = 0 \\
sim_T(\mathcal{A}_{S_2}, \mathcal{A}_T) = (s_1, s_2, s_3) \textit{ with} & s_1 = \frac{1}{3}\cdot(1+1+1) = 1 \\
 & s_2 = \frac{1}{3}\cdot(1+1+1) = 1 \\
 & s_3 = 1
\end{array}$$

We obtain $\mathcal{A}_{S_2} \succ_T \mathcal{A}_{S_1}$ *because* $\mathcal{A}_{S_2} >_S \mathcal{A}_{S_1}$. *Since* $ms(sim_T(\mathcal{A}_{S_2}, \mathcal{A}_T)) = 5 > 1 = min$, $cond_T(\mathcal{A}_{S_2})$ *holds true. Consequently, specialist* PES *would return* $R_{\text{PES}} = \{(id, P_{S_2})\}$. *id and* P_{S_2} *denote the trivial signature match and the positive facts of* $\mathcal{A}_{S_2}$, *respectively. This judgement of* PES *is sensible since the proof of* $\mathcal{A}_{S_2}$ *can be "replayed" without any changes whereas (few) changes are necessary to transform the proof of* $\mathcal{A}_{S_1}$ *into a proof of the target.*

Note however, that also $\mathcal{A}_{S_1}$ *is assigned relatively high similarity values that would, in the absence of* $\mathcal{A}_{S_2}$, *result in the selection of the proof of* $\mathcal{A}_{S_2}$ *for flexible re-enactment.*

6 Experimental Results

As already stated, a learning theorem prover has to be given not only the hard and interesting problems of a domain of interest, but also less challenging problems of various difficulty in order to generate an appropriate case base to tackle the hard problems. Unfortunately, in most publications only the hard problems are reported with perhaps one easy example problem so that the human reader gets an impression of the problem domain.

It would have been easy to use in the following our own sets of problems for our own domains, but then one might argue that we have chosen the problems in such a way that they suit our approach. Therefore we tried to ensure that at least the general problems were provided by someone else and we just had to construct equational formalizations for them. This way we were able to come up with two domains and sets of problems from the TPTP library, version 1.2.0 (see [SSY95]). One domain, the area of groups (GRP domain) could be used without any changes, while for the domain logic calculi (LCL domain) no pure unit-equality axiomatization was given. But such an axiomatization can be constructed, as we will see later. A second problem with the LCL domain is that the problems presented in the TPTP library are not one homogeneous domain, but they constitute several (sub)domains, some of which consist of only one hard problem, again. Therefore we have chosen one of these (sub)domains, namely the CN calculus, for our experiments.

In order to allow the readers to observe and judge the effects our ideas have and how our system compares to other provers, we will provide also data on our best heuristics that do not learn and on the results of OTTER (version 3.0, using the *autonomous mode*, see [Mc94]) for the chosen problems. Our best heuristics (for the two domains) are called AddWeight and Occnest (see [DF94]). AddWeight counts function symbols and variables and selects a fact with minimal count. Occnest counts occurrences and nesting of function symbols with respect to the goal. Since OTTER has won the CADE-13 theorem prover competition ([SS96]) in the category "Unit Equality Problems", we think that a comparison with the current state-of-the-art is thus provided.

6.1 The Setting

We carried out the experiments with our prover DISCOUNT (see [ADF95]) that we extended to include the specialist PES, the expert FlexRE and the case base (with the appropriate changes to the supervisor). DISCOUNT is implemented in C on Unix machines and its basic inference engine is old and relatively slow (and not comparable to the basic inference engine of OTTER, although the various experts of DISCOUNT result in a good performance of the distributed system in general and a good sequential performance in some domains of interest).

In our experiments, we limited each distributed run of DISCOUNT to 3 minutes, while we granted OTTER and the single expert runs a timeout of 10 minutes. Consequently, during the bootstrapping process our learning system can make approximately three attempts in the time that was granted to the other provers (not counting the time when the system was successful, which causes an additional reproduction run and additional computations in order to generate the case quadruple for the case base). The ordering in which the problems were tried out is the lexicographical ordering on the names of the problems.

Specialist PES determines a set of appropriate source problems paired with a suitable signature match (from source to target). Since it is possible that there is an astronomical number of possible signature matches, we limited the number of signature matches to be considered for a certain source problem to 100. In practice this is not a severe restriction because in case there are (much) more than 100 possible signature matches they are essentially equivalent (i.e., they do not have different effects on sim_T). All experiments were performed on SUN Sparc-10 machines. The team runs employed two computing nodes (computers in this case). The length of the first working period was set to 10 seconds.

There are 125 problems in the GRP domain of the TPTP library. A substantial part of the problems of this domain were provided by a mathematician, I. Dahn, who presented the domain as he would have presented it to a human student. Due to this reason, this domain is usable for learning provers.

In the LCL domain we used the problems of the CN calculus (for the concrete names of these problems see Table 3). These problems are problems in the area of condensed

Domain	number of	learning team		OTTER		AddWeight		Occnest	
	problems	# solved	%	# solved	%	# solved	%	# solved	%
GRP	125	113	90	93	74	86	67	91	73
LCL (CN)	24	17	71	11	46	12	50	6	25

Table 1: Comparison learning team vs. OTTER vs. best experts

detachment (cf. [Fu96]). In the TPTP, problems of CN are available in conjunctive clause normal form. The inference rule condensed detachment for the CN calculus that allows for inferring $\sigma(t)$ from $i(s,t)$ and s'—σ being the mgu of s and s', and i being a distinguished binary function symbol—is specified by $\neg P(i(x,y)) \vee \neg P(x) \vee P(y)$. All axioms are given as positive unit clauses $P(t_i)$, and the goal is given as a negative unit clause $\neg P(t)$. In order to tackle these problems with DISCOUNT, we introduce two new constants T and F ("true" and "false"), and rewrite the unit clauses to $P(t_i) = T$ or $P(t) = F$. (The goal is proven if $T = F$ can be inferred.) The rule of condensed detachment is realized with $if(and(P(i(x,y)), P(x)), P(y)) = T$ and the obvious equations specifying if and and ($if(T,x) = x$, $and(T,x) = x, \dots$). (OTTER is also given these equational problems for comparability although it can of course handle the original problem format.)

6.2 The Results

Our main goal was to develop a theorem prover that is able to automatically learn and that is therefore able to prove more problems (without help from the user) than other provers. As Table 1 shows, this goal has definitely been achieved. If we look at the percentage of problems that have been proven, then our new version of DISCOUNT has in both domains a lead of over 15 compared to OTTER and even more compared to the best single experts.

In the following, we will examine the results more closely. Tables 2 and 3 report the results for the problems of the two domains (We omitted in the domain group all problems that can be solved by the conventional experts in less than 20 seconds and that were not used as source problems. Also all problems were omitted that could not be solved by our learning team and the conventional experts. Merely one of the latter problems can be solved by OTTER).

Both tables are organized as follows. The first column states the name of the target problem, the second one the name of the source problem that was detected by PES and used by FlexRE in the successful proof attempt. The third column reports the runtime of the successful proof attempt. The columns 4, 5, and 6 report the runtimes of OTTER, AddWeight, and Occnest. The entry "—" indicates that no proof was found before the timeout.

The clustering of the rows indicate the rounds of the bootstrapping process. So, the first cluster shows the problems that could be solved by the conventional team (which is also indicated by the empty source column). The next cluster reports those problems that could be solved using the problems of the first cluster, and those problems that

target	source	runtime	OTTER	AddWeight	Occnest
190-1	—	4s	2s	24s	3s
191-1	—	3s	2s	24s	2s
179-1	—	12s	—	—	—
169-1	191-1	36s	4s	—	—
169-2	190-1	38s	4s	—	—
179-2	179-1	37s	—	—	—
179-3	179-1	38s	—	—	—
186-1	179-1	41s	—	—	—
186-2	179-1	40s	—	—	—
183-1	179-2	40s	—	—	—
183-2	183-1	42s	—	—	—
183-3	183-1	40s	—	—	—
183-4	183-1	42s	—	—	—
167-3	183-1	129s	—	—	—
167-4	183-1	130s	—	—	—
167-1	167-3	32s	—	—	—
167-2	167-4	35s	—	—	—

Table 2: Experiments in the GRP domain (selection)

already have been solved in the round. The last cluster of Table 3 contains the problems that could not be solved.

Table 2 reports a selection of results for the GRP domain. The results of the learning process in the bootstrapping manner can be clearly observed. For example, by solving problem 179-1 conventionally, DISCOUNT was able to solve in the next round problem 179-2, which was basis for solving problem 183-1 (in the same round). Among the problems that were solvable using 183-1 was 167-3, which was used in the fourth round to solve problem 167-1. Note that the start point of this chain, problem 179-1, could only be solved by a team of experts, which demonstrates the potential of TEAMWORK even without learning experts.

As stated before, our learning team was able to solve all problems that OTTER could solve, except for problem 177-2. But this learning team was able to solve 21 other problems (some of which due to the team of the first round without any learning; those are not mentioned in Table 2) that the autonomous mode of OTTER could not solve within the time limit. Our statistic looks even more impressive, if we omit the "easy problems" (i.e. problems solvable by AddWeight or Occnest within 20 seconds) before computing the percentage rates. Then our learning team solves 59%, OTTER 16%, AddWeight 3%, and Occnest 6%.

Table 3 reports our experiments in the LCL domain for the CN calculus. Here, only two rounds were able to increase the number of problems that could be solved. One chain is problem 047-1 that is used to solve 048-1 which allows for solving 050-1 that then is the source for problem 051-1 (all of these proofs found in the second round).

target	source	runtime	OTTER	AddWeight	Occnest
046-1	—	<1s	<1s	<1s	44s
047-1	—	21s	137s	27s	—
059-1	—	56s	—	55s	—
064-1	—	22s	—	31s	37s
069-1	—	11s	25s	15s	—
048-1	047-1	14s	138s	27s	35s
049-1	047-1	26s	—	80s	63s
050-1	048-1	32s	—	—	—
051-1	050-1	15s	—	—	—
052-1	047-1	98s	245s	304s	—
053-1	048-1	77s	484s	—	279s
055-1	047-1	27s	198s	—	—
056-1	047-1	26s	204s	99s	—
057-1	056-1	23s	203s	289s	—
065-1	064-1	12s	317s	53s	—
066-1	065-1	12s	10s	15s	19s
068-1	069-1	71s	—	—	—
054-1	—	—	—	—	—
058-1	—	—	—	—	—
060-1	—	—	—	—	—
061-1	—	—	—	—	—
062-1	—	—	—	—	—
063-1	—	—	—	—	—
067-1	—	—	—	—	—

Table 3: Experiments in the LCL domain

Note that neither 050-1 nor 051-1 can be solved by the other provers within the time limit.

Besides the number of successes also the runtimes are interesting, because they indicate that the successful runs of our learning team must be much better controlled than the successful runs of OTTER. Since the inference engine of OTTER is much faster than the engine of DISCOUNT we can assume that during the OTTER runs a much larger number of inferences were made. Naturally, it has to be mentioned that this better control is the result of additional computational efforts, namely the effort for proving the source problem and the effort that was spent on the unsuccessful attempts to prove the problem during the previous rounds of the bootstrapping process.

Note that, as in the case of the GRP domain, also in the LCL domain there are several problems that are solved due to the cooperation of experts that do not learn.

In general, our experiments show that our concept of a learning theorem prover clearly outperforms the current conventional theorem provers, if the learning prover is provided with enough "exercise" in the domain of interest it has to work in. In this case even the learning process is accomplished by the prover without help from the user.

7 Conclusion and Future Work

We presented a concept for a learning theorem prover that uses methods from several areas of AI. Based on the TEAMWORK multi-agent architecture for distributed search, we employ case-based reasoning and reactive planning together with concepts for similarity of terms.

The combination of these concepts and techniques resulted in a system that clearly outperformed renowned theorem provers in domains that were presented in a learnable way. This means that, as in the case of a human student, not only the hardest problems are given to the system, but also easy and moderate problems that the system can solve alone and that provide the necessary basis for the learning process. Then, in a kind of bootstrapping process, the system is able to solve more and more harder problems.

Although the system has proven to be very successful, nevertheless most components represent only first ideas with regard to the several areas of AI they are taken from. Already there are other concepts that can be used (for example, learning of characteristic features of a source proof, see [Fu96], or learning the structure of promising terms of a domain, see [DS96]) and that will provide a wider range in the use of learned knowledge.

Consequently, the selection process of the right knowledge has to be modified and improved, both with respect to these new experts and with respect to the combination of these experts to form good teams. Also, in addition to the structuring of the knowledge base that is already provided, it may become necessary to refrain from adding some new problems (and their proofs) that do not represent essential new knowledge. This may be based on an analysis of the proof found for a problem. If a source problem was used and the number of unnecessary facts generated was not much higher than the number of the necessary facts, then this problem obviously represents no necessary addition to the case base.

Furthermore, the selection of good results by the referees can be improved by using learned knowledge, although the basis has to remain the a posteriori analysis of the utility of facts. Finally, an important technique of human problem solving is dividing a problem into easier subproblems. While in first-oder theorem proving with generating provers this technique has so far not proved to be successful due to the missing ability to identify suitable subproblems, proof analysis and learning might suggest sets of suitable subproblems (as it is already the case in inductive theorem proving, see [KW94], [Me95]).

References

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

[ADF95] **Avenhaus, J.; Denzinger, J.; Fuchs, Matt.**: *DISCOUNT: A System For Distributed Equational Deduction*, Proc. 6^{th} RTA, Kaiserslautern, LNCS 914, 1995, pp. 397–402.

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

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

[Bu88] **Bundy, A.**: *The use of explicit plans to guide inductive proofs*, Proc. CADE-9, 1988.

[CMM90] **Conry, S.E.; MacIntosh, D.J.; Meyer, R.A.**: *DARES: A Distributed Automated Reasoning System*, In Proc. AAAI-90, 1990, pp. 78–85.

[De90] **Dershowitz, N.**: *A maximal-Literal Unit Strategy for Horn Clauses*, Proc. 2nd CTRS, Montreal, LNCS 516, 1990, pp. 14–25.

[De95] **Denzinger, J.**: *Knowledge-Based Distributed Search Using Teamwork*, Proc. ICMAS-95, San Francisco, AAAI-Press, 1995, pp. 81-88.

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

[DF96] **Denzinger, J.; Fuchs, D.**: *Referees for Teamwork*, Proc. FLAIRS '96, Key West, FL, USA, 1996.

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

[DK96] **Denzinger, J.; Kronenburg, M.**: *Planning for Distributed Theorem Proving: The Teamwork Approach*, Proc. KI-96 (German annual conference on AI), Dresden, GER, LNAI 1137, 1996, pp. 43–56.

[DS96] **Denzinger, J.; Schulz, S.**: *Recording and Analysing Knowledge-Based Distributed Deduction Processes*, to appear in Journal of Symbolic Computation, 1996.

[Fu95a] **Fuchs, Matt.:** *Learning proof heuristics by adapting parameters*, In Armand Prieditis & Stuart Russell, eds., *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1995, pp. 235–243.

[Fu95b] **Fuchs, Matt.:** *Experiments in the Heuristic Use of Past Proof Experience*, SEKI-Report SR-95-10, University of Kaiserslautern, 1995, obtainable via WWW at `http://www.uni-kl.de/AG-AvenhausMadlener/fuchs.html`

[Fu96] **Fuchs, Matt.:** *Experiments in the Heuristic Use of Past Proof Experience*, Proc. CADE-13, New Brunswick, NJ, USA, LNAI 1104, 1996, pp. 523–537.

[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.

[Ko92] **Kolodner, J.L.:** *An Introduction to Case-Based Reasoning*, Artificial Intelligence Review **6**, 1992, pp. 3–34.

[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*, Tech. rep. ANL-94/6, Argonne National Laboratory, 1994.

[Me95] **Melis, E.:** *A model of analogy-driven proof-plan construction*, Proc. 14th IJCAI, Montreal, AAAI Press, 1995, pp. 182–189.

[SS96] **Sutcliffe, G.; Suttner, C.B.:** *The Design of the CADE-13 ATP System Competition*, Proc. CADE-13, New Brunswick, NJ, USA, LNAI 1104, 1996, pp. 146–160.

[SSY95] **Sutcliffe, G.; Suttner, C.B.; Yemenis, T.:** *The TPTP Problem Library*, Proc. 12th CADE, Nancy, LNAI 814, 1994, pp. 252-266.