Coupling Saturation-Based Provers by Exchanging Positive/Negative Information

Dirk Fuchs*
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: `dfuchs@informatik.uni-kl.de`

Abstract

We examine different possibilities of coupling saturation-based theorem provers by exchanging positive/negative information. We discuss which positive or negative information is well-suited for cooperative theorem proving and show in an abstract way how this information can be used. Based on this study, we introduce a basic model for cooperative theorem proving. We present theoretical results regarding the exchange of positive/negative information as well as practical methods and heuristics that allow for a gain of efficiency in comparison with sequential provers. Finally, we report on experimental studies conducted in the areas condensed detachment, unfailing completion, and superposition.

*The author was supported by the *Deutsche Forschungsgemeinschaft (DFG)*.

1 Introduction

In general, automated theorem proving is based on the solution of search problems which usually comprise huge search spaces. Thus, it is on the one hand necessary to develop fast algorithms and to use a lot of tricks in order to obtain a fast implementation (see, e.g., [HBF96]). On the other hand, however, a skillful control of the search conducted by the prover is especially important to deal with hard search problems. Therefore, for first order theorem proving a lot of different calculi, each of them controllable by various heuristics, have been developed. However, it is problematic in this context to decide which calculus and which heuristic should be employed when tackling a given problem. As a matter of fact, the quality of calculi and heuristics usually fluctuates very much depending on the concrete problem (see [SS96]). Also normally no a priori knowledge which gives us hints on the appropriateness of a certain calculus or heuristic for a given problem is available.

Another main problem is that usually a lot of specialized theorem provers for sub-logics of first order logic exist that are able to solve problems stemming from their specialized logic very efficiently, but cannot deal with problems specified in full first order logic. E.g., the availability of high performance equational provers like WALDMEISTER ([HBF96]) or DISCOUNT ([ADF95]) is somewhat limited although many problems contain a lot of formulae where equality is involved (cf. the well-known problem library TPTP [SSY94]).

The first problem, the lack of knowledge about the quality of certain provers for certain problems could indeed be solved by using competitive versions of different provers. But such a system can at most be as good as the best of its provers. Thus, problems that none of the provers can solve remain unsolved. Therefore, our approach for solving this very problem is the development of *cooperative theorem provers*. We want to achieve that on the one hand some provers using different calculi and heuristics work in parallel and that on the other hand a further gain of efficiency—caused by synergetic effects—is possible. Cooperation may also be the right way to deal with our second problem. This is due to the fact that by cooperation specialized provers can support universal provers.

In this report we want to examine how cooperation between several *saturation-based theorem provers* could look like. Note that this is not a very grave restriction because a lot of provers in first order logic with equality belong to this class (e.g., resolution style provers). We deal with the cooperation of both homogeneous and heterogeneous provers. In this context, we call a set of provers homogeneous if all provers employ the same calculus and differ from each other only in the heuristic they employ. We consider a set of provers to be heterogeneous if the provers employ different calculi and work possibly in different (sub-)logics of first order predicate logic.

In literature one can find some approaches that try to couple homogeneous (see, e.g., [Den95], [FD97]) and heterogeneous provers ([Sut92]). All approaches, however, have in common that the provers only exchange *positive information*, i.e. information that give positive hints, e.g. on the applicability of heuristics or deduced facts for solving the proof problem. Our extension to this is that we also use *negative information*, e.g.

information on facts that do not appear to be needed for a proof. Techniques based on negative information were so far only used in (sequential) tableau-based theorem provers. But we present techniques how to use negative information for the cooperation of saturation-based provers. As we will see, combining the exchange of positive *and* negative information offers various possibilities for a change of experience between different provers which often entails an obvious gain of efficiency.

The report is organized in the following way: At first we introduce the basics of saturation-based theorem provers. In particular, we deal with the three application domains condensed detachment ([Luk70]), unfailing completion ([BDP89]), and superposition ([BG94]). After that, we discuss in section 3 which positive/negative information is well-suited for cooperative proving and how this information can be used. Furthermore, we introduce a basic model of cooperative provers. Section 4 deals in more detail with the exchange of positive information. On the one hand, we discuss theoretical aspects regarding the completeness of the provers. On the other hand we point out some concrete practical techniques. Section 5 deals with the same aspects when exchanging negative information. Results of our experimental studies are presented in section 6. We have experimented with both heterogeneous and homogeneous provers, employing a lot of different calculi. Finally, some conclusions and an outlook at possible future work conclude the report.

2 Basics of Automated Deduction

2.1 Fundamentals

The general problem in automated theorem proving is given as follows: Given a set of facts Ax (axioms), is a further fact λ_G (goal) a logical consequence of the axioms? A fact may be a clause, equation, or a general first or higher-order formula. The definition of "logical consequence" depends heavily on the concrete domain one is interested in.

Commonly, automated theorem provers utilize certain *calculi* for accomplishing the task mentioned above. *Analytic calculi* attempt to recursively break down and transform a goal into sub-goals that can finally be proven immediately with the axioms. *Generating calculi* go the other way by continuously producing logic consequences from Ax until a fact covering the goal appears (but there are also some saturation-based calculi that use the goal in inferences). We shall here concentrate on saturation-based calculi.

Typically a saturation-based calculus contains several inference rules of an inference system $\mathcal{I}$ which can be applied to a set of facts (which represents a certain search state). Expansion inference rules are able to generate new facts from known ones and add these facts to the search state. Contraction inference rules allow for the deletion of facts or replacing facts by other ones, thus contracting the fact base (see, e.g., [Der90]). We write $\Lambda \vdash_{\mathcal{I}} \Lambda'$ if we can derive the set Λ' from Λ through the application of one inference rule. A sequence of sets $(\Lambda_i)_{i \geq 0}$ is called an $\mathcal{I}$-derivation if $\Lambda_i \vdash_{\mathcal{I}} \Lambda_{i+1}$ for all i. In order to solve proof problems by utilizing an inference system $\mathcal{I}$-derivations $Ax = \Lambda_0 \vdash_{\mathcal{I}} \Lambda_1 \vdash_{\mathcal{I}} \ldots$ have to be performed until a fact set Λ_n is derived which contains a fact subsuming the goal. An interesting property of such derivations is the *fairness*. As we will see later, fairness is often necessary for the completeness of a prover.

Definition 2.1 (Persistent Facts, Fairness of an $\mathcal{I}$-derivation)
Let $\mathcal{I}$ be an inference system, let I_e be the set of expanding inference rules. $I_e(M)$ denotes the set of all facts that can be derived from M by applying an inference from I_e to some facts from M. Furthermore, let $\Lambda_0 \vdash_{\mathcal{I}} \Lambda_1 \vdash_{\mathcal{I}} \ldots$ be an $\mathcal{I}$-derivation.

1. We define the set of *persistent facts* by $\Lambda^\infty = \cup_{i \geq 0} \cap_{j \geq i} \Lambda_j$.
2. The $\mathcal{I}$-derivation is called *fair* iff $I_e(\Lambda^\infty) \subseteq \cup_{i \geq 0} \Lambda_i$.

As one can recognize a derivation is fair if all conclusions of generating inferences with persistent facts have been computed. For all calculi described in the following subsections fairness of derivations implies completeness, i.e. each valid proof goal can be proven by performing fair derivations until the goal appears (see, e.g., [Fuc96a] for condensed detachment, [Ave95] for unfailing completion, and [BG94] for superposition based theorem proving).

A common principle to solve proof problems algorithmically with a saturation-based calculus is employed by most systems (algorithm SP: sequential prover): Essentially, a theorem prover maintains either implicitly or explicitly a set $\mathcal{F}^P$ of so-called *potential* or *passive facts* from which it selects and removes one fact λ at a time. After the application of some contraction inference rules on λ, it is put into the set $\mathcal{F}^A$ of *activated facts*, or discarded if it was deleted by a contraction rule (*forward subsumption*). Activated facts are, unlike potential facts, allowed to produce new facts via the application of expanding inference rules. The inferred new facts are put into $\mathcal{F}^P$. We assume the expansion rules to be exhaustively applied on the elements of $\mathcal{F}^A$. Initially, $\mathcal{F}^A = \emptyset$ and $\mathcal{F}^P = Ax$. The indeterministic selection or *activation step* is realized by heuristic means. After the generation of a new fact and before its insertion into the set of passive facts, a heuristic weight $\omega_\lambda \in \mathbb{N}$ is associated with the fact. This heuristic weight remains unchanged during the proof process. It is needed for the activation step since the fact with the smallest heuristic weight is selected. Conflicts are handled using the FIFO strategy.

The heuristic weight of a fact is computed by employing a heuristic $\mathcal{H}$. A heuristic can utilize following information as input: Firstly, a heuristic can use the syntactic structure of a fact, e.g. its number of symbols. Secondly, it can depend on the systems of active and passive facts $\mathcal{F}^A$ and $\mathcal{F}^P$ of the prover at the moment of the generation of λ. E.g., the goal-oriented heuristics defined in [DF94] fall back on this information so as to compute structural differences between a fact and current goals. Finally, a heuristic can also use the *derivation tree* T_λ of a fact λ. The non-root nodes of such a tree T_λ are marked with all facts needed to infer λ, the root is marked with λ. There is an edge from nodes marked with facts $\gamma_1, \ldots, \gamma_n$ to a node marked with γ, if γ is the result of an inference with premises γ_i $(1 \leq i \leq n)$. The edges are marked with the names of the inferences that they represent. Such derivation trees can be used in order to prefer facts that are descendants of certain facts, e.g., the original proof goal. All in all, for a fact λ we obtain $\omega_\lambda = \mathcal{H}(\lambda, \mathcal{F}^A, \mathcal{F}^P, T_\lambda)$.

We call a heuristic *fair* if it guarantees for each start set Λ that each fact being derived from Λ by employing algorithm SP and hence being passive at a certain moment is either activated or discarded after a finite period of time. For a set of facts Λ we call a heuristic Λ-fair if it guarantees that—when starting with input set Λ—each fact being passive at a certain moment is either activated or discarded after a finite period of time. A fair heuristic is Λ-fair for each set Λ. However, a heuristic being Λ-fair for a set Λ is in general not fair.

Theorem 2.1 *There is a heuristic $\mathcal{H}$ that is Λ-fair for a fact set Λ but not fair.*

Proof: Consider the superposition calculus. Let $>= \emptyset$ be the ordering used for superposition. For a clause C, $|C|$ denotes the sum of the numbers of predicate, function, and variable symbols in C. We define a heuristic $\mathcal{H}$—only depending on syntactic properties of a fact—as follows:

$$\mathcal{H}(C) = \begin{cases} 0 & , C \text{ is positive} \\ |C| & , \text{otherwise} \end{cases}$$

Then, for $\Lambda = \{c = d \vee P(f(a)), \neg P(x) \vee P(f(x))\}$ $\mathcal{H}$ is Λ-fair. However, $\mathcal{H}$ is not fair: If the input set is $\Lambda' = \{c = d \vee P(f(a)), \neg P(x) \vee P(f(x)), \neg Q(g(g(a))) \vee R(g(g(a)))\}$, then the last clause remains passive infinitely long. $\square$

It is easy to recognize that the use of fair heuristics implies that the algorithm conducts fair derivations. If the algorithm employs a heuristic which is Λ-fair it produces a fair derivation when starting with $\Lambda_0 = \Lambda$.

2.2 Condensed Detachment

A typical example for saturation-based calculi is the inference system $\mathcal{CD}$ which contains the inference rule *condensed detachment* (`CondDet`) (see [Tar56] and [Luk70] for motivation and a theoretical background). Since $\mathcal{CD}$ contains only one expansion and one contraction inference rule it is very simple. But nevertheless resulting proof problems can be very challenging. Therefore, condensed detachment was chosen as a test domain by several researchers before ([Pet76], [MW92], [Sla93], [Wos95], [Fuc96b]) and the choice of condensed detachment as one test domain surely is justified. The rules of the inference system $\mathcal{CD}$ manipulate first-order terms. These terms are defined as usual, involving a finite set $\mathcal{F}$ of function symbols and an enumerable set of variables $\mathcal{V}$.

`CondDet` in its basic form is defined for a distinguished binary function symbol $f \in \mathcal{F}$. `CondDet` allows to deduce $\sigma(t)$ from two given facts $f(s,t)$ and s' if σ is the most general unifier from s and s'. $\mathcal{CD}$ contains—besides the expanding rule `CondDet`—the contracting rule `Subsum`. This rule allows for the deletion of a fact t if a fact s and a substitution σ exist such that $\sigma(s) \equiv t$. A proof problem $\mathcal{A} = (Ax, \lambda_G)$ is solved if a fact subsuming the goal can be deduced.

For our experimental studies (see section 6) we have employed the theorem prover CoDe as described in [FF97].

2.3 Superposition extended with Sorts

The theorem prover SPASS ([WGR96]) we have chosen to experiment with is an automatic prover for first order logic with equality. It is based on the superposition calculus (see [BG94]). The inference rules of the superposition calculus can be divided into expansion and contraction (also called reduction) rules as we have seen before. The expansion rules (ordered inference rules) contain the common rules of the superposition calculus, i.e. superposition left and right, factoring, equality resolution, and equality factoring. The reduction rules contain well-known rules like subsumption and rewriting. Furthermore, SPASS utilizes additional reduction rules, like the deletion of tautologies and the condensing rule which allows to replace a clause C by $\sigma(C)$ if $\sigma(C) \subset C$. Since SPASS recognizes unary predicates as sorts ([Wei93]) the inference system of SPASS is extended by rules needed for processing sort information. These rules have to be applied to facts before they can be involved in "normal" expansion and contraction rules. Because of the fact that the rules can be regarded as expansion rules we do not distinguish between ordered inference rules and rules that apply to some special sort information in the following.

In this context proof problems are usually given in form of a clause set C that has to be proven inconsistent. But since this has to be done by deriving the empty clause $\Box$ this task is equivalent to the task of solving the proof problem $(C, \Box)$. Thus, in the following we will maintain our notion of a proof problem.

2.4 Completion without Failure

The *unfailing completion* procedure (see [BDP89]) offers possibilities to develop high performance theorem provers (e.g., DISCOUNT [ADF95]) in pure equational logic. In this context the axioms are always universally quantified equations, the proof goal is an arbitrarily quantified equation. The inference system underlying the unfailing completion procedure is in main parts a restricted version of the superposition calculus. It contains one expansion inference rule—the generation of so-called *critical pairs*—that corresponds to the superposition rule. More exactly, if we assume that a special (reduction) ordering $>$ is given, this inference rule is defined as follows: If $s = t$, $u = v$ are equations, $\sigma = mgu(s|p, u)$ exists, $s|p$ is not a variable, $\sigma(t) \not> \sigma(s)$, $\sigma(v) \not> \sigma(u)$, then it is possible to derive the equation $\sigma(t) = \sigma(s)[p \leftarrow \sigma(v)]$. The contraction rules of the unfailing completion procedure correspond to those of the prover SPASS: It is possible to perform rewriting steps, subsume one equation by another, and to delete tautologies.

3 Cooperation in Generating Theorem Provers

Now, we want to introduce general techniques in order to couple different theorem provers by exchanging positive and negative information. Firstly, we deal with the question which kind of positive information is well-suited in order to achieve cooperation between different theorem provers. Furthermore, we show how this information can be utilized. Because of the fact that in literature many different ways exist that describe methods for exchanging positive information we will only sketch this very issue. Secondly, we deal in more detail with the same aspects when employing negative information because the used techniques are novel: we show which negative knowledge may be helpful for different theorem provers and give some hints on how to integrate this knowledge into different provers. Finally, we introduce a basic model of cooperative theorem proving employing positive/negative information.

3.1 Positive Information for Coupling Theorem Provers

As we have mentioned before positive information is everything that gives a prover concrete (positive) hints on how to prove the goal, i.e. information that describe which facts may be helpful or which strategy may be well-suited. Thus, positive information one prover can exchange with others is a set of important lemmas it has generated during the proof attempt so far. Moreover, control information, i.e. information on the quality of certain heuristics, can be exchanged.

The latter kind of information does not seem to be appropriate for distributed cooperative theorem proving. On the one hand it is very difficult to estimate whether a certain heuristic is well-adapted to a certain proof task. Hence, there are only few cooperation models that try to estimate quality of heuristics (see, e.g., [AD93], [Den95]) and this estimation is usually based on vague criteria. On the other hand exchange of information on appropriate strategies requires that the coupled provers are quite homogeneous, i.e. one prover has a strategy as suggested by others at its disposal. Since we are interested in coupling also heterogeneous provers, particularly provers that fall back on different heuristics, we do not cope with the exchange of control information any longer. Instead of the use of control information, positive information that appears well-suited—even when coupling heterogeneous provers—is a set of important facts (lemmas) a certain prover has deduced during the proof attempt so far. Relevancy of these lemmas is determined by the fact whether they seem to be contributing to a proof of the goal or whether they may be helpful for a receiving prover in other ways (see section 4). The importance of such lemmas is that the receiving prover can employ them without proving them again, i.e. they can directly be integrated into the system of the saturation-based prover. Particularly, if the proofs of such lemmas via the heuristic of the receiving prover require a lot of inferences they might be important for it. Because of the fact that lemmas can easily be integrated into the system of every saturation-based prover and hence are well-suited for coupling heterogeneous provers (see section 6) this kind of information was used in several systems before ([Sut92], [Den95], [FD97]). Therefore, we will also use positive information represented by important lemmas. Essentially, we extend the techniques described in [FD97] so as to couple also heterogeneous provers.

3.2 Negative Information for Coupling Theorem Provers

The usage of negative information in order to achieve cooperation between several (saturation-based) theorem provers is much more difficult. Thus, to our knowledge no approaches exist so far that fall back on such kind of information. Contrarily to positive information we consider an information to be negative if it describes which facts ("bad" facts) or which strategy may not be suitable in order to conclude a proof. Therefore, we discuss in the following whether and how information on bad facts or bad heuristics can be utilized.

In contrast to saturation-based provers information on bad facts can easily be used in analytic theorem provers. Note that analytic provers divide a problem into subproblems (subgoals) that share common variables. Generally, the idea is to perform a so-called *failure caching* (see [MIL+97]) that provides information that certain subproblems cannot be solved with certain substitutions, i.e. that certain instances of facts do not contribute to a proof of the initial problem. Due to efficiency reasons usually a so-called *local failure caching* is employed which does not provide negative information suitable for other provers (cp. [AS92]). In principle, however, also a *global failure caching* is imaginable that is able to support other theorem provers. Thus, an exchange of negative information is at least possible although it is normally not sensible due to efficiency reasons.

Because of the fact that a saturation-based theorem prover does not divide the proof problem into various others and works all the time exclusively on the original proof task, such kind of failure caching is impossible. As a matter of fact, if we want to determine in a saturation-based prover whether a certain fact is a bad fact, i.e. is not needed to prove the goal, the proof must already be known. All in all we can say that a semantic classification of facts into good or bad regarding their ability to contribute to solutions of the proof problem does not seem to be sensible in our context. Since we are not able to classify facts into good or bad in a semantic way we choose in the following a more pragmatic approach: Instead of calling facts good or bad for the *proof* of the goal we classify facts as good or bad w.r.t. their usefulness for the *search* for a proof of the goal. Thus, we try to estimate whether facts may be more or less useful for the *process of finding a proof* instead of estimating whether they are *part of a proof*.

We make these ideas more precise: Often there is not only one proof of a goal but a lot of different proofs exist that usually contain different sets of facts needed in them. A proof can be considered to be better than another proof w.r.t. the process of finding it with a given prover if not so many unnecessary inferences are performed and not so many unnecessary facts are generated during the search. Our aim is to find "good" proofs in this sense, i.e. proofs that can be found in a short proof run. From this point of view facts are bad, i.e. not so useful for proving the goal, if they entail a lot of unnecessary inferences and hence possibly a long proof run. As we will describe in section 5 it is possible to design criteria that are rather well-suited for estimating whether facts are bad regarding this concept. Thus, such a pragmatic estimation if a fact is bad is—in contrast to a semantic estimation—a viable approach.

Until now we have given a short description of the way bad facts might be identified. It is unclear, however, how a prover can utilize information on bad facts it has received from other provers.

On the one hand, if such received bad facts are already in the system of active facts of the prover, a *restructuring* of the search state might be the right way. Through such a restructuring it should be achieved that the prover does not work with bad facts in future. Hence our aim is—similarly to the idea of dynamic programming—that paths in the search space whose exploration would lead to high costs are postponed or even neglected. With the help of this technique possibly short proof runs occur if the bad facts are really not needed and a proof can be found quickly when exploring a proof path not employing the bad facts. Nevertheless, it must be guaranteed that eventually all proof paths are explored so as to preserve the completeness of the proving system if a bad fact is really necessary. As described in section 5 even in such a case a restructuring might be sensible: Parts of a proof that are "parallel" to a certain bad fact, i.e. are not based on this fact, can possibly be exploited faster without using the bad fact and then the remaining parts of the proof can be searched for. Thus, a gain of efficiency is also possible in this case. So to say conventional search-guiding heuristics try to find out which facts are needed for a proof, the additional use of restructuring techniques aims furthermore at finding the optimal order in that these facts should be processed.

On the other hand, information on bad facts of other provers can even have a control aspect: These information offer the possibility to tune the search-guiding heuristic in such a way that the activation of bad facts is postponed. Thus, a posteriori knowledge of other provers (detected bad facts) can be transformed into a priori knowledge (construction of a heuristic). Hence, even the exchange of control information via bad facts is possible. Moreover, it appears to be a more viable approach than the explicit exchange of control information (in form of heuristics) as discussed before.

3.3 Basic Model of Cooperative Theorem Proving

Architecture: Our basic model of cooperating theorem provers that exchange positive/negative information can be described as follows: On each computer in a computer network a theorem prover conducts a search for the common proof goal. Either we let only different incarnations of the same prover cooperate—differing from each other only in the search-guiding heuristic they employ for traversing the search space—and hence have a network of homogeneous provers, or we employ different provers (heterogeneous network) which all tackle the same problem. Note that because of the fact that not all provers can deal with problems specified in the same logic (e.g. DISCOUNT's inference mechanism is restricted to equations), we must sometimes transform the input of different provers so as to let the provers work only with facts given in the logic the provers can deal with. For technical details we refer the reader to section 6. We assume that our network of cooperating provers is completely intermeshed, i.e. each prover communicates its good facts (lemmas) and bad facts individually to all other provers.

Proof Process: Since proof problems are usually search problems of tremendous difficulty it is important that each prover has enough time to perform inferences independently and to tackle the problem without permanent interruption by others. Thus, we decided to let the provers work independently for a while and only cooperate periodically. Basically the working scheme of the provers is characterized by certain phases which are similar to the working phases of the TEAMWORK approach (see [Den95]). While the provers try to solve the same problem independently during

so called *working phases* P_w, cooperation takes place during *cooperation phases* P_c. Working phases and cooperation phases alternate each other. Thus, the sequence of phases is $P_w^0, P_c^0, P_w^1, P_c^1, \ldots$.
As we will see in section 4 it is important that each prover performs at least one activation step during a working phase. During a cooperation phase P_c^i each prover performs the following activities:

- transmission of a set of positive lemmas $\mathcal{P}_i$ to each receiving prover i
- transmission of a set of bad facts $\mathcal{N}_i$ to each receiving prover i
- processing of the facts $\lambda \in \mathcal{P} \cup \mathcal{N}$ received from other provers

Thus, in the following we have to deal with the following aspects for both good and bad facts:
Firstly, we have to cope with the question how we can recognize that certain facts belong to the sets $\mathcal{P}_i$ or $\mathcal{N}_i$. As we will discuss later in more detail, we employ selection functions $\varphi_{\mathcal{P}_i}$ and $\varphi_{\mathcal{N}_i}$ that determine sets $\mathcal{P}_i$ and $\mathcal{N}_i$, respectively. Each function $\varphi_{\mathcal{P}_i}$ ($\varphi_{\mathcal{N}_i}$) employs certain criteria that give *evidences* whether a fact λ should belong to $\mathcal{P}_i$ ($\mathcal{N}_i$) or not. Usually we consider a fact λ to be an element of $\mathcal{P}_i$ ($\mathcal{N}_i$) if it is very evident that it belongs to $\mathcal{P}_i$ ($\mathcal{N}_i$) (its evidence exceeds a certain threshold) and it belongs to the (limited) set of facts with the highest evidences. Note that these evidence criteria only give hints that a fact belongs to a certain set $\mathcal{P}_i$ ($\mathcal{N}_i$), but no hints that it is an element of the set $\mathcal{N}_i$ ($\mathcal{P}_i$). If we assume, e.g., that it is not very evident that a fact λ is member of $\mathcal{P}_i$, it is not a hint that it is an element of $\mathcal{N}_i$. In order to answer the question whether the fact λ is an element of $\mathcal{N}_i$ we have to employ criteria that give us evidences for this. As we will see in the following sections we usually employ different criteria for determining whether a fact is an element of $\mathcal{P}_i$ or $\mathcal{N}_i$. We decided to choose such an approach similar to evidence theory (see [Dem68], [Sha76]) because our definitions of being a lemma or a bad fact are not contrary: If, e.g., a fact is not considered to be a bad fact because it is not recognizable that it has complicated the search for the proof goal, we cannot consider it to be a lemma because it does not need to be very important for other provers. Correspondingly, if a fact is not considered to be a lemma we should not classify this fact as a bad fact because it nevertheless may be needed in the proof and the use of the fact may not be very costly. Thus, the concepts of being a lemma or a bad fact are not necessarily contrary. However, it is reasonable that the concepts are at least disjunct (as described shortly in more detail).
Secondly, we have to determine how to process the information received from the other provers. Whereas the positive facts can easily be processed by integrating them into the system of active facts, it is unclear how the processing of bad facts as discussed before—restructuring of the search state and modification of the heuristic—should look like in detail.
Finally, we have to analyze whether our two cooperation concepts are compatible. So it would not be sensible if $\mathcal{P} \cap \mathcal{N} \neq \emptyset$, i.e. facts are considered to be lemmas as well as bad facts. Therefore, we will discuss whether our selection functions $\varphi_{\mathcal{P}}$ and $\varphi_{\mathcal{N}}$ of good and bad facts, respectively, have this consistency property.

4 Cooperation by Exchanging Positive Facts

As we have mentioned before we want to deal in more detail with the concrete realization of the exchange of important lemmas. As already described, to this end it is necessary to identify important lemmas, to transmit them from one prover to all other provers which are part of the network, and to process the information on good facts received from other provers. We start with the description of the selection and transmission techniques. Note that the methods presented in the following are essentially an extension of the techniques described in [FD97] to the case of heterogeneous provers. After that we describe the processing of the received lemmas. Since we can handle received good facts simply as selected potential facts the processing of facts is no practical problem. Nevertheless, we have to examine theoretically whether completeness of a prover can be lost by the periodical "disturbance" of a prover by the others.

4.1 Selection and Transmission of Lemmata

The detection of important lemmas for a receiving prover is the task of so-called *referees*. Such a referee is a pair $(P_{\mathcal{P}}, \varphi_{\mathcal{P}})$ of a *filter predicate* $P_{\mathcal{P}}$ and a *selection function* $\varphi_{\mathcal{P}}$ that are described in the following. When employing a referee the selection of facts during a cooperation phase takes place in the following manner: At first the active facts of the sender are filtered by the filter predicate. Only facts λ for that $P_{\mathcal{P}}(\lambda)$ holds are allowed to pass through the filter and are candidates for a selection. After that, important lemmas are selected from the remaining facts with the help of $\varphi_{\mathcal{P}}$.

4.1.1 Filtering of Facts

The realization of a filter predicate $P_{\mathcal{P}}$ is the main difference to [FD97]. Since we possibly couple heterogeneous provers only such facts should pass through the filter which the receiving prover can integrate into its set of facts. Consider, e.g., that we try to solve a problem in first order-logic with equality employing cooperative versions of SPASS and DISCOUNT. Then it is on the one hand unnecessary to filter facts generated from DISCOUNT because SPASS is able to work with equations. But on the other hand, from the clauses generated by SPASS only unit equations and inequations should pass through the filter because DISCOUNT can only handle equations as new facts and inequations as new goals. Hence, the task of the filter predicate is to prevent facts of a certain logic from being transmitted to other provers which are only able to deduce facts from another logic.

Moreover, the filter predicate $P_{\mathcal{P}}$ can be utilized to additionally limit the set of facts $\varphi_{\mathcal{P}}$ can select from. Typically facts are filtered out that are (thought of as) redundant for the receiving prover. Redundant are all axioms and facts selected in an earlier cooperation phase. Furthermore, heuristic criteria can be used to reduce the number of facts passing through the filter. Thus, it is possible to gain efficiency because the number of facts that have to be judged by $\varphi_{\mathcal{P}}$ is reduced.

4.1.2 Selection of important Lemmata

When selecting good facts two main problems occur: At first we have to determine our system architecture, i.e. we have to decide where the selection of facts should take place and who should be receiver of the facts. Secondly, we must choose criteria for the selection of facts.

As described in [FD97] the following three alternatives for a system architecture are imaginable: The first alternative is to select facts at the sender site (employing a so-called *send-referee*) and to send the set of facts to all other provers. Thus, only one referee is needed. The second possibility is to use send-referees and to select facts at the sender site, too, but to select facts individually for each receiver and to transmit hence to each receiver a different set of facts. In order to do this we need for each receiving prover i an own referee. Finally, it is possible to select facts at the receiver site. Thus, the filtered facts of one prover are sent to all receiving provers and the selection is performed by each receiver (employing a so-called *receive-referee*).

The second main problem we have to solve is to choose criteria for the selection of facts. As we have sketched in section 3 each selection function $\varphi_{\mathcal{P}}$ used for selecting a lemma set $\mathcal{P}$ employs one or more judgment functions which give evidences that a fact is element of $\mathcal{P}$. Such judgment functions ψ_j map the facts to natural numbers. A fact λ is considered to be the better the higher the value $\psi_j(\lambda)$ is. The selection by $\varphi_{\mathcal{P}}$ then takes place in such a manner that a certain number n_j of facts is selected via each function ψ_j.

In general, the different kinds of knowledge that are incorporated into the judgment functions determine both the system architecture and their concrete realization. In general, it is possible to integrate only local knowledge into the judgment functions, i.e. knowledge about the sender. Moreover, also knowledge about the receiver can be integrated into the judgment functions. Knowledge about the receiver can be divided into two different classes: Firstly, the minimum of knowledge one can have is knowledge about the identity of the receiver, i.e. knowledge about its calculus and underlying logic. Secondly, one can have further knowledge about the concrete status of the prover at a certain point in time: This kind of knowledge can again be divided into knowledge about the current search state of the prover (i.e. its active and passive facts) and knowledge about its method to change the search state in future (i.e. its heuristic). Note, that the two kinds of knowledge about the receiver are not strictly separated from each other. So, knowledge about the concrete setting of the receiving prover can only be employed in a reasonable way if also the identity of the receiver is known: If we want to estimate whether a lemma is useful for a prover in a certain situation we must be able to estimate which kinds of inferences the receiver can perform with the lemma and its system of facts. Thus, we must also know the calculus the receiver employs.

Now, if we want to integrate only local knowledge into the judgment functions the selection must take place at the sender site because this kind of knowledge is only available there. Furthermore, it is sufficient to employ only one selection function $\varphi_{\mathcal{P}}$ and to send the selected facts $\mathcal{P}$ to all receivers. The judgment function ψ_S ("statistical judgment")—introduced in [DF96]—judges facts w.r.t. local knowledge. This function counts for each fact λ and for each inference type, the inferences λ was involved in so far. If we assume that λ was involved N_i times in inference i the fact is judged by $\psi_S(\lambda) = \sum_i \alpha_i \cdot N_i$. The α_i rate the inferences in such a way that "good" inferences, i.e. inferences that contract the system of facts, are weighted positive, "bad" inferences (application of expansion rules) are weighted negative.

As we have already mentioned a second kind of knowledge that can be employed is knowledge about the heuristic of the receiver. Selection of facts employing such kind of knowledge is performed at the sender site, too, but we have to select facts individually for each receiver i. This is due to the fact that the heuristics of the receivers differ from each other. Judgment function ψ_H ("heuristical judgment"), described in [FD97], employs knowledge about the heuristic of the receiver. Basically it favors facts that have a small heuristic weight regarding the part of the heuristic of the receiver that is only based on syntactic properties of a fact, but have at least one ancestor with a high heuristic weight. Thus, it is probable that these facts are novel for the receiver.

Finally, it is possible to judge facts regarding the current search state of the receiver which reflects in a certain way its current needs. In order to do that facts have to be selected at the receiver site, i.e. we have to employ our third system architecture. The judgment functions ψ_U and ψ_{SG} utilize knowledge about current needs of a receiver. ψ_U judges quality of facts w.r.t. their possible *usefulness* for reducing future search effort. To this end the function counts for each fact λ how many contraction inferences can be applied to λ and facts from the current system of active facts of the receiver. ψ_{SG} measures the quality of facts w.r.t. their ability to contribute to the solution of certain *(sub-)goals*. For a concrete definition of these functions see [FD97].

The different kinds of knowledge can easily be combined so as to realize more intelligent selection functions. So one can employ both local knowledge and knowledge about the heuristic of the receiver, by employing judgment functions ψ_S and ψ_H for one selection. It is even possible to use additionally knowledge about the current needs of the receivers. Thus, some facts are selected via ψ_S and ψ_H, transmitted to each receiver, and then an additional selection via ψ_U and ψ_{SG} takes place. As discussed in [FD97] it is sufficient to employ only functions ψ_S and ψ_H if good heuristics are at the provers' disposal which generate a lot of important lemmas. However, if the heuristics use only simple syntactic criteria it is wise to employ all kinds of knowledge for selecting facts. Such a selection, however, is more inefficient because a lot of judgment functions have to be applied to the facts.

4.2 Processing of received Lemmata

In this section we will examine theoretical aspects of the processing of lemmas received in a cooperation phase. In the following, we consider a prover whose underlying calculus is complete in a certain logic (e.g. an unfailing completion based prover in pure equational logic or a superposition based prover in full first oder logic with equality). As we have mentioned in section 2 each valid proof goal can be proven by performing a fair derivation until a fact subsuming the goal appears. As we have also seen, our algorithm SP produces fair derivations if it employs a fair heuristic.

If we couple our prover with others by exchanging lemmas its internal algorithm SP must be slightly modified in order to process the received lemmas: It is necessary to activate facts not only from the set of potential facts during the working phases (recall that in each working phase we require at least one real activation, i.e. the insertion of a new fact into $\mathcal{F}^A$), but also periodically lemmas received from other provers must be activated during the cooperation phases. We assume that all received lemmas are activated during a cooperation phase in an arbitrary order. We call this modified algorithm from now on CP (cooperative prover). Now, the question arises whether the prover remains complete. More exactly, we are interested in the question whether it is sufficient to activate facts from the potential facts with a fair heuristic in order to be able to prove every valid goal or whether it is possible that completeness is lost because of the periodical "disturbance" through other provers.

We make this precise: First, it is necessary to extend our notion of $\mathcal{I}$-derivations to $\mathcal{I}_{Dist}$-derivations.

Definition 4.1 (Inference System with Disturbance $\mathcal{I}_{Dist}$)
Let $\mathcal{I}$ be an inference system, let I be the set of inference rules of $\mathcal{I}$. Let Λ_0 be an initial set of facts. If M is a set of facts and A a fact, $M \models A$ denotes that A is a logic consequence of M. Then the *inference system with disturbance* $\mathcal{I}_{Dist}$ contains the inference rules $I_{Dist} = I \cup \{\texttt{Dist}\}$, with

$$(\texttt{Dist}) \qquad \Lambda \vdash \Lambda \cup \{\lambda\}; \Lambda_0 \models \lambda$$

This describes exactly what happens during a cooperation phase: Facts derived from other provers, i.e. logic consequences from the initial system of facts, can be integrated into the system of facts without an additional proof.

A (infinite) sequence of fact sets $(\Lambda_i)_{i\geq 0}$ is an $\mathcal{I}_{Dist}$*-derivation* iff $\Lambda_i \vdash_{\mathcal{I}_{Dist}} \Lambda_{i+1}$, $\forall i \geq 0$. We call an $\mathcal{I}_{Dist}$-derivation $(\Lambda_i)_{i\geq 0}$ fair iff $I_{Dist,e}(\Lambda^\infty) \subseteq \cup_{i\geq 0}\Lambda_i$. (We do not consider `Dist` to be an expanding inference.) Again, regarding our calculi each valid goal can be proven by performing a fair $\mathcal{I}_{Dist}$-derivation until the goal appears. Note that our algorithm CP produces $\mathcal{I}_{Dist}$-derivations. We call the heuristic that CP employs *fair despite disturbance* if it guarantees for each start set Λ that each fact being derived from Λ by employing algorithm CP and hence being passive at a certain moment is either activated or discarded after a finite period of time. The notion of "Λ-fair despite disturbance" is defined in analogy to before. If CP employs a heuristic which is fair despite disturbance it produces only fair $\mathcal{I}_{Dist}$-derivations. If it uses a heuristic which is Λ-fair despite disturbance it produces a fair $\mathcal{I}_{Dist}$-derivation when starting with the set Λ.

Now, we want to examine whether the notions of fairness and fairness despite disturbance are the same. If this is true then each prover which performs fair $\mathcal{I}$-derivations with a fair heuristic would also remain complete if it is coupled with others via the exchange of lemmas. Unfortunately, this is not the case in general. For certain problems it is even possible that completeness gets lost when the heuristic is only based on syntactic properties of a fact. But at least if we employ a *strong fair* heuristic (see [AD93]), i.e. a heuristic where the set $M_z = \{\lambda : \exists \text{ fact sets } \mathcal{C} \text{ and } \mathcal{D}, \text{ and a derivation tree } T_\lambda : \mathcal{H}(\lambda, \mathcal{C}, \mathcal{D}, T_\lambda) = z\}$ is finite for all numbers z, we can preserve completeness.

Theorem 4.1 *Let $\mathcal{I}$ be an inference system, $\mathcal{I}_{Dist}$ be the respective inference system with disturbance. Let Λ be a set of facts. Then it holds:*

1. *Each heuristic which is (Λ-)fair despite disturbance is (Λ-)fair.*
2. *There are heuristics which are fair but not fair despite disturbance.*
3. *There is a fact set Λ and a heuristic $\mathcal{H}$ which is only based on syntactic properties of a fact, such that $\mathcal{H}$ is Λ-fair but not Λ-fair despite disturbance.*
4. *Each strong fair heuristic is fair despite disturbance.*

Proof:

1. This is trivial because $\mathcal{I}$ and SP are restricted versions of $\mathcal{I}_{Dist}$ and CP, respectively.

2. We consider the area of superposition based theorem proving. If a fact λ has been derived, $Anc^*(\lambda)$ denotes the set of all facts needed to infer λ, i.e. the set of all facts in the derivation tree T_λ. $|\lambda|$ denotes the number of symbols in λ. Then, consider the following heuristic $\mathcal{H}$ with

$$\mathcal{H}(\lambda, T_\lambda) = \begin{cases} 0 & , \exists \lambda' \in Anc^*(\lambda) : \lambda' \text{ is result of an inference of type } \texttt{Dist} \\ |\lambda| & , \text{otherwise} \end{cases}$$

If we perform algorithm SP no disturbance takes place, i.e. $\mathcal{H}(\lambda, T_\lambda) = |\lambda|$. Thus, $\mathcal{H}$ is a fair heuristic. However, $\mathcal{H}$ is not fair despite disturbance: Let $>= \emptyset$ be the ordering used for superposition, let the initial set of potential facts be $Ax = \{f(a) = a; f(x) = a \to f(f(x)) = a; f(a) = a, f(f(a)) = a \to f(c) = d\}$. If we first activate $f(a) = a$ and $f(x) = a \to f(f(x)) = a$ (according the heuristic weight) and then a disturbance takes place that adds $f(f(a)) = a$ (a logic consequence from Ax) to the active facts, the infinite sequence of facts $f^{i+2}(a) = a$, $i \geq 0$, is generated and activated. Thus, the fact $f(a) = a, f(f(a)) = a \to f(c) = d$ remains passive infinitely long.

3. We consider again the area of superposition theorem proving. Let $>= \emptyset$ be the ordering used for superposition. For a clause C, $|C|$ denotes the sum of the numbers of predicate, function, and variable symbols in C. $l(C)$ is the number of literals in C. We define a heuristic $\mathcal{H}$—only depending on syntactic properties of a fact—as follows:

$$\mathcal{H}(C) = \begin{cases} 3 \cdot l(C) \dot{-} |C| & , C \text{ is positive} \\ 0 & , C \text{ is negative} \\ |C| & , \text{otherwise} \end{cases}$$

Then, for $\Lambda = \{\neg P(b), P(e), c = d \vee P(f(a)), \neg P(f(x)) \vee P(f(f(x)))\}$ $\mathcal{H}$ is Λ-fair. However, $\mathcal{H}$ is not Λ-fair despite disturbance: Consider the situation that firstly $\neg P(b)$ is activated, after that the facts $c = d \vee P(f(a))$ and $\neg P(f(x)) \vee P(f(f(x)))$ are received from outside and activated immediately. Then, the infinite sequence of facts $c = d \vee P(f^2(a))$, $c = d \vee P(f^3(a))$, $\ldots$ will be activated, i.e. $P(e)$ remains passive infinitely long.

4. Since for each passive fact there are only finitely many facts with a smaller weight, and since in each working phase at least one fact is activated, no fact remains passive infinitely long. □

As we can see it might be the case that a fair heuristic for SP is "not fair for CP" (not fair despite disturbance), i.e. coupling a prover with others can cause incompleteness. Fortunately, most heuristics are strong fair and hence also fair despite disturbance. Nearly all heuristics we used in SPASS and DISCOUNT have this very property.

5 Cooperation by Exchanging Negative Facts

The topic of this section is to describe how to couple different theorem provers by exchanging negative information, i.e. information on bad facts. As described previously we discuss the following two aspects: On the one hand we describe a method for detecting facts that behave badly w.r.t. the principles introduced in section 3. On the other hand we present possibilities in which way a theorem prover can employ received bad facts in order to improve its search.

5.1 Determination of Bad Facts

Analogously to section 4 two steps must be executed: At first the active facts of a prover are filtered by a filter predicate $P_{\mathcal{N}}$ and then some facts are selected from the remaining ones by a selection function $\varphi_{\mathcal{N}}$.

5.1.1 Filtering of Facts

Filtering of facts is again necessary due to the heterogeneity of the cooperating provers. So, it is convenient to let only such facts pass through the filter that the receiving prover can use in further inferences. If we want, e.g., to couple SPASS and DISCOUNT we have to filter the non-unit clauses generated from SPASS because DISCOUNT cannot cope with information on clauses C, $|C| > 1$. Moreover, such a filter predicate is used to avoid that a certain fact is sent twice to receiving provers.

Finally, in addition to this further (heuristic) criteria might be useful to limit the number of facts that pass through the filter. Hence the number of facts that have to be judged by time-consuming selection criteria can be reduced. For our experiments it was sufficient to resign further filtering criteria. However, it might be the case, e.g. if a lot of facts are generated by a prover in a working phase, that a further filtering of facts is reasonable.

5.1.2 Selection of Bad Facts

Whereas the process of filtering facts is very similar to section 4, the selection via function $\varphi_{\mathcal{N}}$ differs from the selection via $\varphi_{\mathcal{P}}$. Principally the selection appears to be analogous to before: So, again three architecture models are imaginable: Selecting facts for all receivers at the sender site, selecting facts individually for each receiver at the sender site (hence we need for each receiver i an individual selection function $\varphi_{\mathcal{N}_i}$), and selecting facts—from the set of facts of the sender that pass through the filter—at the receiver site. Moreover, each selection function $\varphi_{\mathcal{N}}$ employs certain judgment functions ψ_j that give evidences whether a fact may be a bad fact. Again, a fact λ is considered to be the worse the higher the value $\psi_j(\lambda)$ is.

But, in contrast to before only two different kinds of knowledge can reasonably be employed for selecting facts: As we will discuss in the following, local knowledge can as before be used for selecting bad facts. Integration of knowledge about the receiver into the selection process, however, does not appear to be very sensible. Only knowledge about the heuristic of the receiver might be useful for selecting bad facts whereas knowledge about the current search state of the receiver cannot be used efficiently. Thus, we employed only the first two architecture models, i.e. we determined bad facts only at the sender site. We discuss the question whether different kinds of knowledge are useful for selecting bad facts in more detail:

First, we can recognize that mainly local knowledge, i.e. knowledge about the sender, is useful for detecting bad facts. Whereas it was possible to take the individual situation of the receiver into account beyond local knowledge when selecting positive lemmas[1], it does not seem to be sensible to do so when selecting bad facts. This is due to the fact that our definition of being a bad fact is a local definition from the point of view of the prover that has generated the fact: As defined, a fact is a bad fact if it complicates the search for the proof of the goal. As a matter of fact, this can only be estimated by performing a retrospective view on the past and to check whether a fact was often involved in time-consuming expanding inferences. Thus, bad facts have to be detected employing local knowledge and must hence be determined at the sender site because the needed information is only available there. Furthermore, it is sensible to send to all other provers the same selected facts.

From a more pragmatic point of view, however, it is possible to employ—in addition to local knowledge—knowledge about the heuristic of the receiver. The idea behind this is that an information on a bad fact is definitely useless if the receiving provers traverse the search space in directions completely different from that of the sender. In such a case it is not very probable that the receivers will use the information on bad facts because they will never activate or even generate such facts. When employing knowledge about the heuristic of the receiver it is possible to estimate whether the receiver is possibly able to activate such detected bad facts and should therefore be informed on them. With the help of this knowledge we can hence better distinguish between a lot of bad facts: If a lot of bad facts have the same quality according to local criteria it is wise to choose such facts the receiver will possibly activate. A selection as described that falls back on knowledge about the heuristic of the receivers can take place at the sender site, too, but it has to be performed individually for each receiving prover.

Employing knowledge on the concrete system of facts of a receiving prover is not reasonable in our context. Regarding only the system of facts of a prover it is not

[1] Note that an important quality criterion of a lemma was its estimated usefulness for the receivers.

possible to determine whether bad facts might be activated in future. Indeed one can decide—employing knowledge about the system of the receiver *and* knowledge about its heuristic—if a bad fact will be activated within a fixed number of steps of the prover. But this is very inefficient since we have to perform deduction steps in order to decide this. Because of this inefficiency and because of the fact that it is not so dramatic to send a receiver a little information it is not able to use we decided to resign this kind of knowledge.

In our concrete realization we decided to employ only one kind of judgment functions ψ_B for judging facts and integrated both kinds of knowledge in it local knowledge and knowledge about the heuristic of the receiver. Thus we selected bad facts at the sender site individually for each receiver. The concrete realization of ψ_B is as follows: We split ψ_B in a part $P_{\mathcal{L}}$ based on local knowledge and a part $P_{\mathcal{H}}$ based on knowledge about the heuristic of the receiver. Thus, we obtain $\psi_B(\lambda) = P_{\mathcal{L}}(\lambda) + P_{\mathcal{H}}(\lambda)$.

At first we consider local knowledge in order to estimate if a fact contributes with a high probability only to long proof runs, i.e. proof runs where a lot of unnecessary facts are generated and a lot of unnecessary inferences are performed. Naturally, we can only decide this if we have found the proof of the goal. Nevertheless, there is a simple criterion we can already utilize during the search for the proof: Facts being involved in a lot of expanding inferences but only in a few contracting inferences contribute with a high probability to long proof runs. This is mainly because of the fact that they generate a lot of offspring. Since usually a lot of this offspring is not needed for the proof (but often used for inferences) the prover is forced to waste a lot of computation time for handling such facts. Hence, facts that possibly contribute to long proof runs can be detected by counting the inferences they were involved in. The concrete realization of $P_{\mathcal{L}}$ is as follows:

Definition 5.1 (Local part $P_{\mathcal{L}}$)
Let $\mathcal{I}_1, \ldots, \mathcal{I}_n$ be the kinds of inferences a theorem prover can perform, let $\mathcal{I}_1, \ldots, \mathcal{I}_k$ be the expanding, $\mathcal{I}_{k+1}, \ldots, \mathcal{I}_n$ be the contracting inferences. For each fact λ and each inference type $j \in \{1, \ldots, n\}$ let $|\mathcal{I}_j(\lambda)|$ be the number of inferences of type j the fact λ took part in so far. Moreover, let $\alpha_1, \ldots, \alpha_n$ be real numbers, $\alpha_i \geq 0$. Then

$$P_{\mathcal{L}}(\lambda) = \sum_{i=1}^{k} \alpha_i \cdot |\mathcal{I}_i(\lambda)| - \sum_{i=k+1}^{n} \alpha_i \cdot |\mathcal{I}_i(\lambda)|$$

Obviously, facts are judged the better the higher their number of contracting inferences and the lower their number of expanding inferences is.

The second part $P_{\mathcal{H}}$ is simply given as $P_{\mathcal{H}} = -\mathcal{H}_R$, $\mathcal{H}_R$ is the part of the heuristic of the receiving prover that only depends on the syntactic structure of a fact. Hence, facts that have a high weight regarding the heuristic of a receiver are considered to be less negative because it is quite improbable that the receiver has activated them or will activate them in future.

5.2 Processing of Bad Facts

In general we process the set of bad facts received from other theorem provers in two different manners: On the one hand we try to modify the heuristic weights of pasive facts of the receiving prover so as to postpone or even avoid the activation of such bad facts (if they are not already activated). On the other hand we try to perform a restructuring of the search state so as to delay the investigation of certain proof paths.

5.2.1 Modification of Heuristic Weights

A modification of the heuristic weights of a prover regarding a set $\mathcal{N}^j$ of bad facts received in cooperation phase P_c^j is only sensible if some facts $\mathcal{N}_P^j \subseteq \mathcal{N}^j$ are not already activated. The aim of such a modification is then to postpone or avoid the activation of facts $\lambda \in \mathcal{N}_P^j$. Thus, it is possible to convert a posteriori knowledge of the sender (detected bad facts) into a priori knowledge (heuristic weights) of the receiver.

In order to realize such a modification a simple memory based approach is imaginable:

Definition 5.2 (Modified Heuristic Weights $\widehat{\omega_\lambda}$)
Let γ be a real-valued parameter, $\mathcal{N}^j$ be the set of all bad facts the prover has obtained in cooperation phase P_c^j ($j \geq 0$). Then the modified heuristic weights $\widehat{\omega_\lambda}$ of passive facts λ (with original heuristic weights ω_λ) equal in each working phase P_w^i the following weights ω_λ^i: If $i = 0$ then $\omega_\lambda^0 = \omega_\lambda$. If $i > 0$ then

$$\omega_\lambda^i = \begin{cases} \gamma \cdot \omega_\lambda & , \lambda \in \mathcal{N}^{i-1} \\ \omega_\lambda & , \textit{otherwise} \end{cases}$$

The parameter γ determines whether the activation of bad facts is completely impossible during the next working phase (strong penalty) or whether the activation is only delayed (moderate penalty). We utilized the setting $\gamma = 2$, i.e. quite a strong penalty.

Finally, we want to answer the question which influence the modification of the heuristic weights has on the completeness of theorem provers. In particular, we want to examine if a prover that originally uses a fair heuristic or a heuristic which is fair despite disturbance might become incomplete when modifying the heuristic weights as described. Under certain restrictions on the original heuristic of a prover this is impossible:

Theorem 5.1 *Let $\mathcal{I}_{Dist}$ be an inference system with disturbance. Let $\mathcal{H}$ be a strong fair heuristic. Furthermore, let CP be realized in such a way that it uses $\mathcal{H}$ and modifies heuristic weights $\widehat{\omega_\lambda}$ of passive facts λ as described in the preceding definition. Then CP produces only fair $\mathcal{I}_{Dist}$-derivations.*

Proof: Recall that by the modification of the heuristic weights in cooperation phase P_c^i the weight of a bad fact $\lambda \in \mathcal{F}^{\mathcal{P}}$ is changed only for the next working phase P_w^{i+1}. Moreover, no prover sends a bad fact λ twice to a receiver. This way and because of the fact that only finitely many provers work in our network the heuristic weight of a bad fact is only changed in finitely many working periods. Hence, there is an index k

such that the heuristic weight of λ remains unchanged in all working phases P_w^j, $j \geq k$. Because of the fact that in each phase at least one fact is activated and that only a finite number of facts exist whose heuristic weight is lower than ω_λ, λ will be selected in a certain working phase. Therefore, CP conducts still fair $\mathcal{I}_{Dist}$-derivations. $\Box$

Thus, a prover cannot become incomplete if it starts with a strong fair heuristic.

5.2.2 Restructuring of the Search State

By restructuring the search state we want to achieve that a prover does not work with already activated bad facts or at least postpone inferences where bad facts take part in. Hence we try to correct a posteriori a wrong decision we made due to our low a priori knowledge (poorly performing heuristic guiding).

Generally, such a kind of restructuring is—as already mentioned—only reasonable if bad facts are already in the system of active facts of the receiver. (Note, that the modification of the heuristic weights prevents at least temporary that the receiver works with bad facts that are still passive.) However, such a kind of restricted application of the restructuring technique would entail some problems: Remember that we want to combine the exchange of positive and negative information among homogeneous and heterogeneous provers. Now, it is well-known (see, e.g., [DF96]) that methods based on the exchange of important lemmas between homogeneous provers require the heuristics of the cooperating provers to be quite different. Therefore, in our context we have to employ different heuristics, too. But this entails that different provers do not have so many active facts in common. Moreover, this is also true when coupling heterogeneous provers. Thus, restructuring of the search state could not be employed very often because in the most cases bad facts as suggested by the sender are not in the system of active facts of the receiver. Nevertheless, surely a lot of other unnecessary facts are in the systems of active facts of the receivers. Thus, it would be desirable for a prover to derive from the badness of certain facts of other provers that also some of its activated facts are possibly unnecessary.

The solution of this problem is to consider a kind of "analogy": According to our definition of badness a fact is bad if it complicates the search for the proof goal. Since this often depends on certain syntactical properties of a fact we could infer from the *syntactical similarity* between a fact $\tilde{\lambda}$ and a bad fact λ that also $\tilde{\lambda}$ is a bad fact. Regarding our definition of the judgment function ψ_B one can see that facts λ and $\tilde{\lambda}$ that are syntactically very similar would have with a high probability similar judgments $\psi(\lambda)$ and $\psi(\tilde{\lambda})$ (if the facts were activated at the same time). Thus, it is surely justified to generalize the information on bad facts to syntactically similar facts $\mathcal{S}$. So, λ represents a *scheme* $\mathcal{S}$ of bad facts.

There are different ways to express a scheme of facts that are all in a way similar regarding their syntactic properties. It is possible, e.g., to use as a scheme of facts the set of all facts that share the same *feature values* (see, e.g., [Fuc96b]). Features describe syntactical properties of facts by means of natural numbers. Another possibility is to represent fact schemes via so-called *term arity trees* (see [DS96]). All these alternatives abstract strongly from concrete facts. We, however, want to orient ourselves on the concrete facts received from the other provers. Therefore, we chose a different way to generalize facts to fact schemes by using an explicit measure for similarity defined on terms. In our approach a scheme $\mathcal{S}_\lambda$ of facts w.r.t. a certain fact λ and a set $\mathcal{F}^{\mathcal{A}}_{\mathcal{R}}$ of active facts of a receiver $\mathcal{R}$ is given as

$$\mathcal{S}_\lambda = \{\lambda' \in \mathcal{F}^{\mathcal{A}}_{\mathcal{R}} : sim(\lambda, \lambda') \geq min_{sim}\}$$

Usually, we restrict the size of $\mathcal{S}_\lambda$ to a fixed number n_{max}. The function sim regulates the similarity of facts and hence the degree of generalization. Thus, the degree of abstraction can be controlled very flexibly.

Surely, a lot of different realizations of the function sim are imaginable. Our approach is to allow for small deviations from the syntactical structure of a fact. More exactly, we consider two facts to be similar if one differs from the other only in some variable positions (see below). Furthermore, we allow for exchanging function symbols for others provided that they have the same arity. Thus, we can expect that facts which are similar in such a manner are probably involved in the same kinds of inferences. Hence, we can expect that they probably would have corresponding values w.r.t. ψ_B if they were activated at the same time. In order to give an exact definition of our function sim we need a few preliminary definitions. Note that we only try to generalize unit clauses because in our experiments we restricted us to this case. (Non units as bad facts can anyhow only occur if we couple two instantiations of SPASS: CoDe and Discount only work with units, thus non units can either not occur or are filtered out.) Furthermore, we consider only unit clauses to be possibly similar to other units. Now, we write for two terms $t \approx_{EQ} t'$ if t differs from t' only at variable positions, i.e.

$$t \approx_{EQ} t' \text{ iff } \begin{cases} t \equiv f(t_1, \ldots, t_n), t' \equiv f(s_1, \ldots, s_n), t_i \approx_{EQ} s_i \\ t \equiv x \in \mathcal{V}, t' \equiv y \in \mathcal{V} \end{cases}$$

$\#_{EQ}(t, t')$ counts the different variable positions if $t \approx_{EQ} t'$, otherwise it is equal to $|t|$. Because of the fact that bounded variables can be renamed, $\#_{EQ}(t, t')$ counts the minimum of all respective numbers for t and $\sigma(t')$, σ is a renaming of bounded variables in t'.

The relation $\approx_{strEQ}$ is defined as follows:

$$t \approx_{strEQ} t' \text{ iff } \begin{cases} t \equiv f(t_1, \ldots, t_n), t' \equiv g(s_1, \ldots, s_n), t_i \approx_{strEQ} s_i \\ t \equiv x \in \mathcal{V}, t' \equiv y \in \mathcal{V} \end{cases}$$

Again, $\#_{strEQ}(t, t')$ counts the minimum of different function symbols and variables w.r.t. certain renamings of variables if $t \approx_{strEQ} t'$. For terms t and t' not being in $\approx_{strEQ}$-relation $\#_{strEQ}(t, t') = |t|$. Our similarity measure sim employs the relations $\approx_{EQ}$ and $\approx_{strEQ}$ in order to measure the structural difference of facts. For literals (that do not contain the equality predicate) t and t' (which one can consider to be terms) it holds

$$\boxed{sim(t, t') = \max(p_1, \frac{1}{2} p_2)}$$

where p_1 is given as $p_1 = \left(\frac{|t| - \#_{EQ}(t,t')}{|t|}\right)$. p_2 is defined analogously to p_1, employing $\approx_{strEQ}$ and $\#_{strEQ}(t, t')$ instead of $\approx_{EQ}$ and $\#_{EQ}(t, t')$, respectively. As one can see the similarity between two literals is the higher the smaller their syntactical deviations are. If we try to generalize equations we use a slightly modified notion of similarity. The similarity between a literal λ and an equation $\lambda' \equiv s = t$ is measured by

$$\boxed{sim(\lambda, \lambda') = \max(sim(\lambda, s = t), sim(\lambda, t = s))}$$

The similarity between two equations $\lambda \equiv s = t$ and $\lambda' \equiv u = v$ is given by

$$\boxed{sim(\lambda, \lambda') = \max\left(\frac{1}{2}(sim(s,u) + sim(t,v)), \frac{1}{2}(sim(s,v) + sim(t,u))\right)}$$

Note that our similarity measure is rather restrictive, i.e. for most different facts λ and λ' we have $sim(\lambda, \lambda') = 0$. For our experimental evaluation sim was sufficient. But there might be the situation where the measure should be refined.

Since we are now able to generalize information on bad facts we do not try to restructure our search state w.r.t. the received bad facts $\mathcal{N}_i$ but we use the set $\mathcal{N}_{\mathcal{S},i} = \cup_{\lambda \in \mathcal{N}_i} \mathcal{S}_\lambda$. Note that we limited in our experiments the size of each $\mathcal{S}_\lambda$ to the value 1. It is to be emphasized that all facts $\lambda \in \mathcal{N}_{\mathcal{S},i}$ are in the system of active facts of the receiver.

A restructuring of the search state employing the set $\mathcal{N}_{\mathcal{S},i}$ should now guarantee that the examination of search paths is postponed where facts $\lambda \in \mathcal{N}_{\mathcal{S},i}$ are involved in. In order to do this one can use the so-called *inference rights* as described in [Fuc97]. Inference rights are essentially annotations to facts that describe rights to take part in certain inferences. Then, we do not employ any longer algorithm SP (if we do not employ information on lemmas) or algorithm CP (if we exchange lemmas) using the conventional inference systems $\mathcal{I}$ or $\mathcal{I}_{Dist}$, respectively, but we employ algorithms $SP^{\mathcal{R}}$ or $CP^{\mathcal{R}}$ employing inference systems $\mathcal{I}^{\mathcal{R}}$ or $\mathcal{I}^{\mathcal{R}}_{Dist}$ working on facts with rights (λ, R). Such a right R is a set of inference types λ can be involved in. The inference system $\mathcal{I}^{\mathcal{R}}$ ($\mathcal{I}^{\mathcal{R}}_{Dist}$) contains the rules of $\mathcal{I}$ ($\mathcal{I}_{Dist}$) but considers the rights in such a way that an inference can only performed if all facts involved in it have the respective inference right. Additionally, $\mathcal{I}^{\mathcal{R}}$ ($\mathcal{I}^{\mathcal{R}}_{Dist}$) contains a rule for detracting rights. Then, by utilizing inference rights such a postponing as suggested is realized in $SP^{\mathcal{R}}$ ($CP^{\mathcal{R}}$): Expanding inferences with facts $\lambda \in \mathcal{N}_{\mathcal{S},i}$ are forbidden for a while by detracting the inference rights for all expanding inferences ("deactivation") in the cooperation phase. Thus, the examination of proof paths where such facts are involved in is postponed. Moreover, we do not need to perform so many time-consuming expanding inferences and are able to investigate other proof paths with less computation effort. Note that by utilizing such a fine-grained control via inference rights we do not loose simplification power because contraction inferences are still allowed. Thus, we can indeed find proofs very quickly if the "deactivated" facts are really not needed for any proof. Nevertheless, it might be the case that such a detected bad fact is necessary for the proof. Thus, we employ a *recover mechanism* for inference rights (by using an additional recover rule in $\mathcal{I}^{\mathcal{R}}$ ($\mathcal{I}^{\mathcal{R}}_{Dist}$)) that can give inference rights back to a fact and performs all inferences that were postponed by deactivation. Also in this case it is possible that we gain efficiency: Parts of the proof that are "parallel" to the bad fact, i.e. the facts in this part do not have the bad fact as an ancestor, can possibly be found faster if the bad fact is deactivated. The remaining parts of the proof can be found after the recovery of rights. Thus, the restructuring of the search state can entail a rearrangement of the proof parts found by the prover and thus a gain of efficiency. Such a recovery of rights is performed by $SP^{\mathcal{R}}$ ($CP^{\mathcal{R}}$) periodically in the cooperation phases.

Finally, we want to examine if a prover that performs fair $\mathcal{I}$-derivations or fair $\mathcal{I}_{Dist}$-derivations with a strong fair heuristic might become incomplete when considering the received information on bad facts for restructuring the search state. First, we define fairness of $\mathcal{I}^{\mathcal{R}}$- ($\mathcal{I}^{\mathcal{R}}_{Dist}$-)derivations analogously to section 2, i.e. all expansion inferences must be performed to persistent facts without considering the inference right. Again, fairness implies in our context the completeness of the provers. Then following theorem holds true:

Theorem 5.2 *Let $\mathcal{I}$ ($\mathcal{I}_{Dist}$) be an inference system (with disturbance). Let $\mathcal{H}$ be a strong fair heuristic. Furthermore, let $SP^{\mathcal{R}}$ ($CP^{\mathcal{R}}$) be realized in such a way that it uses $\mathcal{H}$, that each deactivated fact can recover all inference rights after a finite period of time, and that no facts are deactivated infinitely often. Then $SP^{\mathcal{R}}$ ($CP^{\mathcal{R}}$) produces only fair $\mathcal{I}^{\mathcal{R}}$- ($\mathcal{I}^{\mathcal{R}}_{Dist}$-)derivations.*

Proof: We have to show that by performing inferences with $SP^{\mathcal{R}}$ ($CP^{\mathcal{R}}$) and inference system $\mathcal{I}^{\mathcal{R}}$ ($\mathcal{I}^{\mathcal{R}}_{Dist}$) finally all expanding inferences are performed to persistent facts.

Let λ be an arbitrary persistent fact. At first we show that there is a certain moment τ where λ is either activated or recovers its inference rights and for all moments $\tau' \geq \tau$ holds: λ remains active with all rights. Because of the fact that we employ a heuristic which is strong fair we know that λ will be activated at a certain moment. If λ will never be deactivated it persists throughout the search with all inference rights. Otherwise, λ will be deactivated at some moments. Because of our precondition we know that there is a moment where λ recovers all inference rights and does not loose these rights any longer.

Now, we can prove our theorem: Let λ be again an arbitrary persistent fact, let τ be the moment where λ is either activated or recovers its inference rights so that for all moments $\tau' \geq \tau$ holds: λ remains active with all rights. Because of the fact that all expanding inferences are performed to λ and active facts at the moment τ and because of the fact that expanding inferences are exhaustively applied to active facts at each activation or recover step, all expanding inferences employing λ and any other persistent facts will finally be performed. $\square$

Hence, weak and easily realizable conditions on the heuristic and the algorithm can guarantee completeness.

5.3 Consistency of our Cooperation Concepts

As we have mentioned before we call the combination of our two cooperation concepts consistent if $\mathcal{P} \cap \mathcal{N} = \emptyset$, i.e. a receiver does not obtain contradictory information on the quality of facts. This property is important in practice: If we assume that a fact $\lambda \in \mathcal{P} \cap \mathcal{N}$ exists, then the lemma λ would not be used by the receiver because the processing of bad facts avoids this. Thus, a lemma would be sent to the receiver that it will not be able to use (at least for a long period of time). Hence, we would waste communication amount and computation time for the processing of the fact without any gain.

Technically, this problem can be solved by discarding all facts $\lambda \in \mathcal{P} \cap \mathcal{N}$. This would anyhow reduce the computation time for the processing. More desirable, however, is that consistency is given without discarding, i.e. $\mathcal{P} \cap \mathcal{N} = \emptyset$ holds every time. This is not only desirable due to efficiency reasons but also regarding our judgment functions: It would not be sensible to assume that a fact is on the one hand a good lemma for other provers and on the other hand a fact one should not work with. Therefore, we will now briefly examine whether our judgment functions have this consistency property.

Because of the fact that we have only one judgment function ψ_B for bad facts we must investigate if the judgments of ψ_B and the judgments of the functions ψ_S and ψ_H are different.[2] At first we compare ψ_S and ψ_B: As one can see these functions are rather contrarily defined, i.e. facts that are judged with a high value w.r.t. ψ_S are judged with a small value w.r.t. ψ_B, and vice versa. ψ_S and ψ_B will therefore with a high probability not select the same facts. More problematic is that both ψ_H and ψ_B, prefer facts having a small weight according to the heuristic of the receiver. But note that the numerical bigger part of ψ_B is (usually) not the heuristic part $P_{\mathcal{H}}$ but the local part $P_{\mathcal{L}}$. Moreover, ψ_H takes into account whether a fact λ has an ancestor with a high heuristic weight which is not a criterion when employing ψ_B. Finally, we have to remember that fixed numbers $n_{\mathcal{P}}$ and $n_{\mathcal{N}}$ of good and bad facts, respectively, are selected. In our experiments (see section 6) we used numbers $n_{\mathcal{P}}$ and $n_{\mathcal{N}}$ that were quite small in comparison with the number of facts a theorem prover activates in a working phase. Thus, facts are usually not considered to be good as well as bad. During our experiments in three different domains we could not observe inconsistency.

6 Experimental Results

In the following, we want to examine the potential of our cooperation concepts. In order to do this we conducted our experimental studies in different domains of the well-known problem library TPTP (see [SSY94]) version 1.2.1. We experimented on the one hand with cooperating homogeneous provers, i.e. provers that are based on the same calculus and differ from each other only in the heuristic they use for activating facts. On the other hand, we let also heterogeneous provers employing different calculi cooperate. The two central topics of our empirical examination are: Firstly, we want to evaluate our two cooperation concepts separately and use either only positive information or negative information. Secondly, we cope with the compatibility of our cooperation concepts and let the provers cooperate by both the exchange of important lemmas and bad facts. We start with the homogeneous case and conclude our examination with the heterogeneous case.

[2]Note that we do not take care of the judgments of ψ_U or ψ_{SG}: These functions are either not applied or they work on facts selected with ψ_S and ψ_H.

6.1 Cooperation of Homogeneous Provers

We examined cooperation of homogeneous provers by two case studies: In the area of condensed detachment we coupled instances of the prover CoDe ([FF97]), in the area of superposition based theorem proving instances of the prover SPASS ([WGR96]).

6.1.1 Experiments with CoDe

The test problems this examination is based on stem from experiments performed by McCune and Wos (see [MW92]) with Otter ([McC94]). The problems can also be found in the TPTP library, version 1.2.1, namely in the LCL domain. We use the names the problems have been given in the TPTP.

In order to exchange important lemmas we used the following settings: Since CoDe is able to employ very powerful heuristics (see [Fuc96b]) based on learned knowledge it is very probable that a lot of important lemmas are in the set of active facts of each prover. Therefore it is—as already mentioned—not necessary to integrate very much knowledge into the selection of facts, but it is possible to construe powerful referees that are based only on local knowledge and knowledge about the heuristic of the receiver. Thus, we decided to employ only the judgment functions ψ_S and ψ_H for selecting facts at the sender site and performed no additional selection at the receiver site. Hence, we could select facts very efficiently. We parameterized function ψ_S in such a manner that the number of contracting inferences was rated by the value 1, the number of expanding inferences by -1. We selected in each cooperation phase the fixed number of 15 lemmas. Note that this number was much smaller than the number of facts each prover could activate during a working phase.

In order to select bad facts we employed a filtering with $P_{\mathcal{N}}$ and a selection with $\varphi_{\mathcal{N}}$ with the help of function ψ_B. The filtering was performed mainly as described in section 5, i.e. axioms and facts that were selected in an earlier cooperation phase were not allowed to pass through the filter. Additionally, facts λ could not pass through the filter if their number of deactivations exceeded the threshold 10 so as to preserve completeness. ψ_B was parameterized in such a way that all ratings α_i were set to the value 1. We selected in each cooperation phase 30 bad facts. Again, this number was much smaller than the number of facts activated in a working phase.

Since we are interested in exploiting the potential of our two cooperation concepts we compare for each problem our system of cooperating provers with the best heuristic for the respective problem which we could find. It is to be emphasized, that these heuristics are usually based on learned knowledge and are therefore very well-adapted to the proof problems. In order to allow for a fair comparison one of the coupled incarnations of CoDe activated facts with the best heuristic for the problem. In our experiments we coupled this heuristic with an incarnation of CoDe using a "standard" heuristic. Such a standard heuristic either orients itself on simple syntactic criteria—e.g. it counts the number of function symbols and variables in a fact—or it is a slight modification of goal oriented heuristics introduced in [DF94]. In general, we chose this second heuristic in such a way that it had a rather "opposite behavior" in comparison

problem	ϖ	best heuristic	pos	neg	pos/neg	OTTER
LCL002-1	–	58.5	21.7	39.7	16.3	516
LCL003-1	–	91.4	55.5	77.3	47.2	449
LCL017-1	51.3	51.3	42.7	47.3	38.7	281
LCL040-1	–	14.2	8.0	12.1	7.1	16
LCL054-1	–	34.7	19.3	30.5	17.2	–
LCL058-1	52.9	52.9	17.6	43.7	17.0	423
LCL060-1	56.1	36.9	6.5	34.1	6.7	447
LCL061-1	–	446.6	283.2	428.8	181.2	–
LCL071-1	–	29.1	4.8	29.0	4.3	511
LCL085-1	1200.5	110.3	110.6	39.4	42.1	2172
LCL097-1	44.0	40.5	8.8	40.5	9.2	2
LCL114-1	357.1	25.1	8.4	12.3	6.6	2035
LCL116-1	–	32.8	20.8	31.7	22.3	2041
LCL119-1	–	128.9	100.7	86.7	76.3	362

Table 1: Coupling incarnations of CoDe by exchanging positive/negative information

with the first heuristic. This means that the second heuristic activated only a small number of facts activated by the first heuristic, too.

Table 1 presents an excerpt from our results in the area of condensed detachment. It displays in column 1 the name of the proof problem as it was given in the TPTP, in columns 2 and 3 the run times of a standard heuristic ϖ—employing syntactic criteria—and of the best heuristic, in columns 4–6 the results when coupling our two heuristics by the exchange of important lemmas (entry 'pos'), bad facts ('neg'), or by exchanging both kinds of information ('pos/neg'). All run times are given in seconds and were measured on one or two SPARCstations ELC. In order to point out that we are dealing with hard problems we note in column 7 the run times needed when using OTTER which is perhaps the only serious competitor of CoDe for such kinds of problems (cf. [FF97]). Note that OTTER was not used in its autonomous mode but we depicted here the results published in [MW92] that were achieved by utilizing one of up to six different heuristics. The tests with OTTER were conducted on a SPARCstation 1+, a machine comparable to ours. An entry "–" in table 1 denotes that the respective problem could not be solved within four hours.

If we take a closer look at the results we achieved when coupling two incarnations of CoDe we can recognize the following: By the exchange of positive facts we can achieve satisfactory speed-ups: In a lot of cases the speed-ups are greater than 2, moreover there are some examples where the speed-ups range in an interval from 3 to 6. If we couple the provers by exchanging negative information the results are worse: The speed-ups are lower in comparison with the exchange of positive facts and are generally lower than 2. In nearly all cases, however, also the exchange of negative information increases the power of the proof system. Moreover, there are examples (`LCL085-1`, `LCL119-1`) where we can gain efficiency whereas we cannot speed-up the prover by exchanging lemmas. The results achieved by combining both exchanging of important lemmas and bad facts, outperform significantly the respective results of the best heuristic. Furthermore, for almost all problems the combination of both kinds of information is better than using only one kind alone. For the remaining problems the run times are only slightly longer than the run times of the "best" cooperation concept. Concluding, we can say that cooperation by exchanging positive facts (important lemmas) is able to significantly improve our condensed detachment prover CoDe, and that even the achieved good results can be improved when additionally combining this cooperation technique with the use of negative information.

6.1.2 Experiments with SPASS

We performed experiments with SPASS in order to show that our cooperation concept is not only able to improve very specialized theorem provers but that it is also well-suited for provers that work in full first-order logic. As our test set we chose problems stemming from the CADE-13 ATP system competition ([SS96]). Among these problems we selected those from the categories "unit equality" and "mixed".

A minimal requirement in order to couple different incarnations of a prover is that it has different heuristics at its disposal. Furthermore, it is desirable that these heuristics differ from each other very much because this facilitates the cooperation by exchanging lemmas. SPASS, however, does not employ different heuristics but uses only one fixed strategy. Thus, we were forced to vary this fixed heuristic, that simply counts a weighted sum of the number of variables and two times the number of function symbols in a clause, in order to get different ones. More exactly, we allowed a weighting individually set for each function symbol instead of weighting each occurrence of a function symbol with the value 2. Therewith we can construe a lot of heuristics but it is clear that they do not differ very much w.r.t. the facts that they activate. Thus, it is necessary to select important lemmas very carefully and we decided hence to integrate very much knowledge into the referees. We selected facts with a send-referee and the help of the functions ψ_S and ψ_H at the sender site and performed an additional selection at the receiver site with ψ_U and ψ_{SG}. Concretely, 25 facts were selected from the send-referee and from these facts 10 were selected at the receiver site. As one can recognize these numbers are much smaller than before because SPASS—due to its more complex inference system—does not activate so many facts in a working phase as CoDe.

In order to detect bad facts we used the filter predicate $P_{\mathcal{N}}$ and selection function $\varphi_{\mathcal{N}}$ as described in the preceding section. Note that we did not allow non unit clauses to pass through the filter. We selected in each cooperation phase 5 bad facts which was the best value in our experimental studies.

In order to investigate the potential of our cooperation concepts we compare them with the standard version of SPASS. Again, we let two heuristics cooperate. One of the heuristics was the SPASS standard heuristic, the other was a heuristic generated in the following manner: Function symbols occurring in the axiomatization but not in the conjecture clauses were weighted with the value 5, the other symbols with the value 2.

problem	SPASS	$\mathcal{H}_1$	$\mathcal{H}_2$	pos	neg	pos/neg	OTTER
`LCL196-1`	292.4	292.4	311.7	83.8	272.1	80.2	34.9
`LCL163-1`	10.0	10.0	11.9	7.5	9.8	7.0	–
`GRP048-2`	23.3	23.3	17.4	8.7	12.9	8.3	101.9
`GRP148-1`	951.2	307.9	253.1	184.7	97.4	91.1	70.3
`GRP169-1`	80.1	15.7	29.2	13.7	9.9	10.8	9.9
`GRP169-2`	56.1	56.1	26.2	9.7	14.7	12.8	9.5
`GRP174-1`	8.0	8.0	8.3	5.8	3.9	4.0	9.6
`RNG018-6`	639.9	639.9	199.7	152.3	87.1	79.1	0.5
`NUM009-1`	8.1	8.1	6.3	2.6	8.7	4.2	21.5

Table 2: Coupling incarnations of SPASS by exchanging positive/negative information

Only for examples `GRP169-1` and `GRP148-1` we deviated from this method. Because of the fact that for these problems the standard heuristic has only a weak performance we employed instead of the standard heuristic another heuristic that weighted certain function symbols not occurring in the goal clauses with the value 5.

Table 2 displays an excerpt of our results. We list here only the results of such problems the standard heuristic of SPASS was able to solve. Moreover, we omitted problems SPASS could solve within 5 seconds (all run times were achieved on one or two SPARCstations-20). Again, column 1 shows the name of the problem, column 2 the run time needed when using the standard version of SPASS. Columns 3 and 4 display the run times needed by the two coupled heuristics, columns 5–7 the results when employing our cooperation concepts. Finally, column 8 presents the results when using OTTER. These results are not so interesting because SPASS and OTTER behave very differently when solving certain examples. So, SPASS performs very well on certain problems and OTTER poorly, and vice versa for other problems. Table 2 shows that OTTER works mostly better than SPASS on our test set, but there is a problem OTTER is not able to solve within 1000 seconds. We noted the results of OTTER merely in order to show that we do not deal with trivial problems. More important, however, is the comparison of the single versions of SPASS with our cooperative provers.

The results reveal that cooperation by exchanging important lemmas allows also in superposition based theorem proving for an obvious gain of efficiency in comparison with the standard version of the prover. Also the speed-ups w.r.t. the average of the coupled heuristics, i.e. the speed-ups we achieve if we compare our results with a random choice of one of our heuristics, are satisfactory. Moreover, our cooperative provers solve the most problems about two times faster compared with the best of the coupled heuristics. It is to be emphasized that due to our inefficient implementation (exchange of facts via files) the results can further be improved. Coupling of superposition provers by using negative information does not entail results as good as before: we achieve on the average a speed-up of 1.6 compared with the best of the coupled heuristics. But similarly to the area of condensed detachment there are difficult examples (`GRP148-1`, `RNG018-6`) where the exchange of negative information leads to much better results than the exchange of positive information. Employing both kinds of information entails in the prevailing number of cases a further gain of efficiency. The medium speed-up compared with the best of the coupled heuristics is greater than 2. Furthermore, particularly when tackling difficult problems (`LCL196-1`, `GRP148-1`, `RNG018-6`) the results are very encouraging. This gives us hope that our concept is especially well-suited when dealing with hard problems. All in all we can say that the two cooperation concepts allows also in this application domain for a satisfactory gain of efficiency. Furthermore, both concepts are compatible.

6.2 Cooperation of Heterogeneous Provers

We examine cooperation among heterogeneous provers by coupling SPASS and DISCOUNT. If we tackle problems specified in pure equational logic the situation does not seem to be very different from the preceding section: SPASS and DISCOUNT are complete in equational logic and perform principally the unfailing completion algorithm. However, DISCOUNT is especially optimized for equational logic and can perform inferences because of its simple fact representation faster than SPASS. Moreover, DISCOUNT has many goal oriented heuristics (see [DF94]) at its disposal which make it particularly suited for problems specified in equational logic. Therefore, when coupling SPASS and DISCOUNT one can expect even in equational logic a different behavior as when coupling two different incarnations of SPASS.

If we are interested in solving problems in full first order logic with equality we have to take care of the problem that DISCOUNT can only cope with unit clauses.[3] This is not a problem during the proof run because facts DISCOUNT cannot work with are not allowed to pass through the filter of SPASS. But, we must at least transform the initial axiomatization. Our solution of this problem is following decomposition of a proof problem, represented as a set of clauses $\mathcal{C}$ whose inconsistency should be shown: Each positive equation $P(t_1, \ldots, t_n) = true$ or $s = t$ of $\mathcal{C}$ is chosen as an axiom for DISCOUNT, each negative equation acts as a proof goal. We only considered examples where we could isolate enough positive equations such that the completion of DISCOUNT did not stop. In the case that no negative equation was an element of $\mathcal{C}$ DISCOUNT worked without a proof goal, i.e. in a completion mode. Obviously, by this kind of problem decomposition DISCOUNT is in the most cases not able to solve its proof problem. However, it is able to produce a lot of important lemmas or bad facts from the part of the search space it can traverse.

Nevertheless, we have noticed that it would often be desirable if DISCOUNT was able to work with horn clauses. Thus, we allowed horn clauses to pass through the filter of SPASS and integrated during a cooperation phase descendants of these clauses into the active facts of DISCOUNT with the following simple method: We computed for each horn clause $A_1, \ldots, A_n \to C$ the set $hyper(A_1, \ldots, A_n \to C, \mathcal{F}^A)$ of all positive hyper-resolvents from the horn clause and DISCOUNT's actual system of facts. Note, that this set contains only positive equations. Then, we discarded the horn clauses and processed the set of generated equations analogously to received unit clauses.

[3]Note that unit literals $P(t_1, \ldots, t_n)$ can easily be transformed into equations $P(t_1, \ldots, t_n) = true$.

In order to select important lemmas we used methods similar to those when coupling incarnations of SPASS: We selected 10 facts via ψ_S and ψ_H at the sender site, then an additional selection using ψ_U and ψ_{SG} took place at the receiver site. Finally, a maximum of 5 facts was integrated into the systems of the provers during a cooperation phase. The same number of bad facts was exchanged during each cooperation phase.

Generally, we let SPASS work in its standard mode. Only if SPASS was not able to solve the problem (within 1000 seconds), we experimented with our self-implemented heuristics in order to find a heuristic able to solve the respective problem. However, for all problems presented here where we could not find a proof employing the standard heuristic we could not find a proof with another heuristic, either. DISCOUNT activated facts with a goal-oriented heuristic as described in [DF94]. In order to measure the strength of our cooperation concepts we experimented in the light of various problems taken from TPTP. We dealt particularly with the domains ROB and HEN and present in table 3 a small excerpt from these experiments, enriched with some highlights taken from other domains. Again, problem names can be found in column 1, the best results of SPASS when working alone in column 2. Column 3 displays the run times when using DISCOUNT. In general, the entry "–" denotes that the problem could not be solved within 1000 seconds. Note that DISCOUNT is in a lot of cases not able to solve the problem because it has only parts of the initial proof problem as input. Columns 4–6 show the run times when employing one or both of our cooperation concepts, column 7 the run time when using OTTER. Again, we listed these times merely in order to show that we do not deal with trivial problems. We do not believe that a comparison with OTTER is really sensible due to its different behavior compared with SPASS and DISCOUNT. More interesting is a comparison of the results of the sequential versions of SPASS and DISCOUNT with the results they can achieve if they cooperate.

If we take a closer look at the results from table 3 we can observe that they are very promising and much better than in the homogeneous case. By exchanging important lemmas we are able to solve all of the listed problems, whereas SPASS is only able to solve 71%, DISCOUNT 18%, and OTTER 71%. Even when we take into account the fact that SPASS *and* DISCOUNT work as cooperative prover there are still problems (`GRP177-2`, `GRP179-1`, `LDA010-2`) neither SPASS nor DISCOUNT can cope with, but that can be solved if the provers cooperate. Moreover, we achieve at other problems high speed-ups up to the value 20 (`BOO007-4`). Nevertheless, there are problems where we achieved only a small gain of efficiency (e.g. `LCL163-1`). However, at the most difficult problems (run time greater than 100 seconds) the speed-ups are satisfactory.

When coupling provers only by exchanging negative information the speed-ups are lower but in almost all cases we could gain efficiency. Moreover, there are also some examples (`GRP169-1`, `ROB005-1`, `ROB008-1`, `ROB016-1`, `HEN009-5`) where the speed-ups are rather high. As we have already emphasized in the preceding experiments we can find problems (`ROB016-1`, `HEN009-5`, `LCL163-1`) where the use of negative information outperforms the use of positive information. Very hard problems, however, seem to be out of reach if we restrict us to negative information: Problems neither SPASS nor DISCOUNT were able to solve remain unsolved. Thus, we can again observe that the potential of the exchange of positive information in order to improve coupled provers appears to be higher.

problem	SPASS	DISCOUNT	pos	neg	pos/neg	OTTER
BOO007-4	403.4	–	19.3	373.4	19.3	10.9
GRP169-1	80.1	–	41.9	50.5	40.3	9.1
GRP177-2	–	–	58.3	–	58.3	–
GRP179-1	–	–	80.7	–	80.7	–
LCL143-1	16.1	–	1.7	6.8	1.3	1.1
LCL163-1	10.0	12.0	7.7	7.4	6.3	–
ROB005-1	–	109.6	39.9	74.2	35.2	44.9
ROB008-1	–	98.8	33.3	58.4	17.0	0.4
ROB011-1	105.3	–	21.9	82.4	20.4	0.6
ROB016-1	9.8	–	4.6	3.8	3.4	0.7
ROB022-1	15.1	–	6.9	12.2	4.7	5.0
ROB023-1	204.6	–	4.4	204.8	4.1	2.2
LDA010-2	–	–	682.7	–	682.7	–
HEN009-5	309.9	–	105.9	99.4	109.3	119.7
HEN010-5	68.7	–	14.7	64.8	19.6	48.6
HEN011-5	41.2	–	26.2	41.5	23.3	–
CIV001-1	24.9	–	12.6	20.4	12.6	7.4

Table 3: Coupling SPASS and DISCOUNT by exchanging positive/negative information

Combining both exchange of important lemmas and bad facts, is again the best way for coupling theorem provers. In the prevailing number of cases the results are better than only using positive information. Indeed, the gain is often low but sometimes conspicuous enhancements are possible (ROB008-1).

All in all we can ascertain that cooperation of heterogeneous provers is a very encouraging method, particularly when dealing with hard problems. The use of positive information leads to satisfactory results, the use of negative information should be further enhanced. Employing both kinds of information is a very good approach for establishing cooperation among different (saturation-based) theorem provers.

7 Conclusion and Future Work

We have presented two different cooperation techniques well-suited for coupling homogeneous and heterogeneous provers: On the one hand the exchange of positive information realized by exchanging important lemmas. On the other hand the exchange of negative information, i.e. certain "bad facts".

We examined these two cooperation concepts theoretically and could observe that the completeness of a prover can get lost if we allow for an uncontrolled exchange and processing of positive/negative information. Fortunately, we discovered weak conditions on the heuristic of a prover and the processing of the received information that are sufficient for completeness. Thus, in most cases our cooperation concepts do not destroy completeness. Moreover, we pointed out practical ways to realize cooperative provers and introduced some concrete techniques and heuristics. The experimental results have shown that our methods indeed enabled the cooperative prover to clearly outperform sequential provers. Particularly in the heterogeneous case the results were very satisfactory because we were able to solve problems that none of the coupled provers could solve when working alone.

It is clear that the experimental studies should be extended. Essentially, however, there are two main aspects that should be further examined:

On the one hand it would be interesting to examine whether other positive or negative information than good or bad facts might be used for coupling different provers. As we have already mentioned it appears to be difficult to use, e.g., control information in order to couple different provers. Nevertheless, it would be interesting to perform further research to find out other different kinds of positive and negative information one can employ. On the other hand it would be desirable to extend our presented methods also to the case of analytic provers. Further research should hence deal with the problem to find out if at least our abstract concepts of coupling theorem provers by exchanging positive/negative information can be transformed to tableau-style provers.

References

[AD93] J. Avenhaus and J. Denzinger. Distributing equational theorem proving. In *Proc. 5th RTA*, pages 62–76, Montreal, 1993. LNCS 690.

[ADF95] J. Avenhaus, J. Denzinger, and M. Fuchs. DISCOUNT: A System For Distributed Equational Deduction. In *Proc. 6th RTA*, pages 397–402, Kaiserslautern, 1995. LNCS 914.

[AS92] O.L. Astrachan and M.E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In *Proceedings of CADE-11*, pages 224–238, Saratoga Springs, USA, 1992. Springer LNAI 607.

[Ave95] J. Avenhaus. *Reduktionssysteme.* Springer, 1995.

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

[BG94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[Dem68] A.P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society, Series B*, 30:205–247, 1968.

[Den95] J. Denzinger. Knowledge-based distributed search using teamwork. In *Proc. ICMAS-95*, pages 81–88, San Francisco, 1995. AAAI-Press.

[Der90] N. Dershowitz. A maximal-literal unit strategy for horn clauses. In *Proc. 2nd CTRS*, pages 14–25, Montreal, 1990. LNCS 516.

[DF94] J. Denzinger and M. Fuchs. Goal oriented equational theorem proving. In *Proc. 18th KI-94*, pages 343–354, Saarbrücken, 1994. LNAI 861.

[DF96] J. Denzinger and D. Fuchs. Referees for teamwork. In *Proc. FLAIRS '96*, pages 454–458, Key West, 1996.

[DS96] J. Denzinger and S. Schulz. Learning domain knowledge to improve theorem proving. In *Proc. CADE-13*, pages 62–76, New Brunswick, 1996. LNAI 1104.

[FD97] D. Fuchs and J. Denzinger. Cooperation in theorem proving by loosely coupled heuristics. Technical Report SR-97-03, University of Kaiserslautern, Kaiserslautern, 1997.

[FF97] D. Fuchs and M. Fuchs. CoDe: A powerful prover for problems of condensed detachment. In *Proc. CADE-14*, pages 260–263, Townsville, 1997. LNAI 1245.

[Fuc96a] D. Fuchs. Inference rights for controlling search in generating theorem provers. Technical Report SR-96-12, University of Kaiserslautern, Kaiserslautern, 1996.

[Fuc96b] M. Fuchs. Experiments in the heuristic use of past proof experience. In *Proc. CADE-13*, pages 523–537, New Brunswick, 1996. LNAI 1104.

[Fuc97] D. Fuchs. Inference Rights for Controlling Search in Generating Theorem Provers. In *Proc. EPIA '97*, pages 25–36, Coimbra, 1997. LNAI 1323.

[HBF96] T. Hillenbrand, A. Buch, and R. Fettig. On Gaining Efficiency in Completion-Based Theorem Proving. In *Proc. RTA-96*, pages 432–435. Springer LNCS 1103, 1996.

[Łuk70] J. Łukasiewicz. *Selected Works.* L. Borkowski (ed.), North-Holland, 1970.

[McC94] W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, 1994.

[MIL$^+$97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *special issue of the Journal of Automated Reasoning*, 1997.

[MW92] W. McCune and L. Wos. Experiments in Automated Deduction with Condensed Detachment. In *Proc. CADE 11*, pages 209–223, Saratoga Springs, 1992. LNAI 607.

[Pet76] G. J. Peterson. An automatic theorem prover for substitution and detachment systems. *Notre Dame Journal of Formal Logic*, 19(1):119–122, 1976.

[Sha76] G. Shafer. *Mathematical Theory of Evidence.* Princeton University Press, Princeton, 1976.

[Sla93] J. Slaney. Scott: A model-guided theorem prover. In *Proc. IJCAI '93*, pages 109–114, Chambery, 1993.

[SS96] G. Sutcliffe and C. Suttner. ATP System Competition held on August 1 in conjunction with the Conference on Automated Deduction (CADE-13). New Brunswick, 1996. Competition results available via WWW at the URL `http://wwwjessen.informatik.tu-muenchen.de/~tptp/CASC-13`.

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *CADE-12*, pages 252–266, Nancy, 1994. LNAI 814.

[Sut92] G. Sutcliffe. A heterogeneous parallel deduction system. In *Proc. FGCS'92 Workshop W3*, 1992.

[Tar56] A. Tarski. *Logic, Semantics, Metamathematics.* Oxford University Press, 1956.

[Wei93] C. Weidenbach. Extending the resolution method with sorts. In *Proc. IJ-CAI '93*, pages 60–65, Chambery, 1993.

[WGR96] C. Weidenbach, B. Gaede, and G. Rock. Spass & Flotter Version 0.42. In *Proc. CADE-13*, pages 141–145, New Brunswick, 1996. LNAI 1104.

[Wos95] L. Wos. Searching for circles of pure proofs. *Journal of Automated Reasoning*, 15:279–315, 1995.