Knowledge-based Cooperation between Theorem Provers by TECHS

Dirk Fuchs, Jörg Denzinger
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: {dfuchs|denzinge}@informatik.uni-kl.de

Abstract

We present a methodology for coupling several saturation-based theorem provers (running on different computers). The methodology is well-suited for realizing cooperation between different incarnations of one basic prover. Moreover, also different heterogeneous provers –that differ from each other in the calculus and in the heuristic they employ– can be coupled. Cooperation between the different provers is achieved by periodically interchanging clauses which are selected by so-called referees. We present theoretic results regarding the completeness of the system of cooperating provers as well as describe concrete heuristics for designing referees. Furthermore, we report on two experimental studies performed with homogeneous and heterogeneous provers in the areas superposition and unfailing completion. The results reveal that the occurring synergetic effects lead to a significant improvement of performance.

1 Introduction

Although the aim of the CADE theorem prover competitions is to find the best automated theorem prover (regarding a given set of problems) a deeper analysis of the results reveals that each prover has different strengths and weaknesses. Naturally, users interested in applying theorem provers to their problems are interested in a system having all strengths of these (and some more) provers and none of the weaknesses. In addition, proving problems should be very efficient and may also make use of clusters of workstations or PCs that are the typical hardware environment these days.

In order to obtain such a system it might be sensible to analyze all known provers, develop concepts and structures that can be used to implement all of their methods, develop heuristics that decide when to use which method, add procedures to divide problems into "digestible" parts for the methods, and design a complex interaction scheme allowing for the distribution of the proof process to several computers. During the years needed to undertake this effort, the developers have to hope that no new ideas resulting in new provers or major updates of existing ones come up, because this entails that they must probably start all over again.

Another solution is to develop concepts that allow the *existing* provers to *cooperate* when solving a proof problem. Such a concept should be easy to integrate into the provers, it should allow the interchange of interesting results only, and it should be open so as to allow adding new provers at any time. Naturally, the provers can run in parallel on different machines.

Henceforth, we present such a concept, our TECHS approach for a knowledge-based cooperation between search agents employing different search models and methods. The general idea of TECHS is to let the provers periodically interchange selected results, i.e. formulas. Formulas can be interchanged via files which may be slow but allows an easy integration of the concept into existing provers, since most of the provers can read data from and write data to files.

The selection process is performed by referees that are known from the TEAMWORK method ([Den95]). But in contrast to TEAMWORK's referees that are totally success-driven in their selection, TECHS employs both success-driven and demand-driven referees. Thus, each connection between provers can utilize a send-referee and a receive-referee that use different knowledge for accomplishing their job. The selection of send-referees is based on the success formulas have had with respect to the search of the sending prover. They can also use some general knowledge about the receiver like its preferences. Receive-referees select formulas from the formulas sent by the send-referees to their prover. Their selection tries to meet the actual demands of their prover.

We demonstrate the usefulness of our approach with two case studies. Firstly, we employed TECHS to let incarnations of the prover SPASS cooperate (thus forming a homogeneous team). This case study showed that despite the slow communication synergetical speed-ups can be observed (with a prover not built for distribution and cooperation). Secondly, we used TECHS to let SPASS cooperate with DISCOUNT (thus forming a heterogeneous team). Besides synergetical speed-ups we also observed that the cooperation resulted in the solution of several problems that none of the provers could solve alone.

This paper is organized as follows: In section 2 we briefly introduce the basics of saturation-based provers. In section 3 we present our TECHS approach. In sections 4 and 5 we describe our experiments with homogeneous and heterogeneous teams. Finally, in section 6 we conclude with some remarks regarding other known cooperation concepts for provers and our future goals.

2 Basics of Automated Deduction

In general, automated theorem proving (in first-order logic) deals with the problem to show the inconsistency of a clause set $\mathcal{M}$. Commonly, automated theorem provers utilize certain *calculi* for accomplishing this. *Saturation-based calculi* continuously produce logic consequences of $\mathcal{M}$ until the empty clause is derived.

Typically a saturation-based calculus is based on an inference system that contains several inference rules which can be applied to a set of clauses (which represents a certain search state). *Expansion* inference rules are able to generate new clauses from known ones and add these clauses to the search state. *Contraction* inference rules allow for the deletion of clauses or replacing clauses by other "simpler" ones. For sets of clauses $\mathcal{M}$ and $\mathcal{N}$, $\mathcal{M} \vdash \mathcal{N}$ denotes that it is possible to derive $\mathcal{N}$ from $\mathcal{M}$ by applying one inference rule.

Usually, a theorem prover which employs a certain calculus maintains either implicitly or explicitly a set $\mathcal{F}^P$ of so-called *potential* or *passive clauses* from which it selects and removes one clause C at a time. After the application of some contraction inference rules on C, it is put into the set $\mathcal{F}^A$ of *activated clauses*, or discarded if it was deleted by a contraction rule (*forward subsumption*). Activated clauses are, unlike potential clauses, allowed to produce new clauses via the application of expanding inference rules. The inferred new clauses are put into $\mathcal{F}^P$. We assume the expansion rules to be exhaustively applied to 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. To this end, a heuristic $\mathcal{H}$ associates a natural number $\mathcal{H}(C) \in \mathbb{N}$ with each $C \in \mathcal{F}^P$. Subsequently, that $C \in \mathcal{F}^P$ with the smallest *weight* $\mathcal{H}(C)$ is selected.

An important property of heuristics is their *fairness*: A heuristic is called fair if it selects clauses in such a manner that no clause stays in $\mathcal{F}^P$ infinitely long. If a prover is based on a complete calculus and it uses a fair heuristic every unsatisfiable proof problem can be solved by deriving clauses with the above algorithm until the empty clause appears.

For our experiments with the TECHS approach we used the saturation-based calculi *superposition* and *unfailing completion*.

In the area of superposition-based theorem proving we conducted experimental studies with the prover SPASS (see [WGR96]). This is an automatic prover for first-order logic with equality. It is based on the superposition calculus (see [BG94]). The unfailing completion procedure (see [HR87], [BDP89]) offers possibilities to develop high-performance theorem provers (e.g., DISCOUNT [ADF95]) in pure equational logic. The inference system underlying the unfailing completion procedure is in main parts a restricted version of the superposition calculus and contains additional reduction rules.

The search state of an equational prover is usually divided into orientable equations, so-called *rules*, and other non-orientable equations.

3 The TECHS approach

The general idea of the TECHS approach (TEams for Cooperative Heterogeneous Search) is to interchange selected clauses between heterogeneous provers in regular time intervals. Firstly, we shall introduce the main principles of the TECHS methodology for realizing cooperating provers. Secondly, we point out methods for exchanging clauses between provers that are part of the cooperative system. Especially, we describe the components responsible for the exchange of clauses, so-called *send-* and *receive-referees*. Finally, we explain possibilities of initializing a team of heterogeneous provers.

3.1 Basics of TECHS

The TECHS approach requires several different provers running in parallel on different computing nodes. Despite the fact that our experiments will only cover saturation-based provers also analytical provers can be used. All provers tackle the same proof problem (although provers may obtain merely parts of it, see below). In our context proof problems are specified in first-order logic (with equality). The provers employ either calculi which are complete for first-order logic (*universal provers*) or calculi which are complete for a sub-logic of first-order logic (*specialized provers*). We assume that unique numbers are assigned to the provers.

As already mentioned, the provers exchange information in regular time intervals. Thus, the working scheme of each prover is characterized by certain phases (similar to the TEAMWORK approach). The sequence of phases is $P_{init}, P_w^0, P_c^0, P_w^1, P_c^1, \ldots$, i.e. after an initialization phase (P_{init}), working (P_w^i) and cooperation phases (P_c^i) alternate each other.

If the original clause set to be refuted is $\mathcal{M}$, each prover i obtains in the initialization phase an initial clause set $\gamma_i(\mathcal{M}) \subseteq \mathcal{M}$ (see section 3.3). Moreover, it gets a schedule of cooperation phases. In the working phases each prover works on its set of clauses employing its inference rules.

In a cooperation phase, the provers exchange some of their generated clauses (see section 3.2). Obviously, a prover should not communicate all new clauses it has generated since the last cooperation phase to all other provers. This would force a receiving prover to process all these clauses (i.e to perform all possible inferences involving them). Moreover, since these clauses persist in the search state they might slow down the prover because they can be used in many inferences in future. Then it is quite probable that the cooperation of the provers does not result in much gain. Furthermore, it is wise to send only a small number of clauses so as to decrease the amount of communication. Hence, we decided to exchange only a subset of the clauses which is selected by using *referees*. Let $\mathcal{F}_i^{A,j}$ be the set of active clauses of prover i at the beginning of cooperation phase P_c^j. Then, the activities in cooperation phase P_c^j are: At first send-referees

examine the active clauses of prover i and determine a set of active clauses $S_{i,k}^j \subset \mathcal{F}_i^{A,j}$ for all receiving provers $k \neq i$. Then, each set $S_{i,k}^j$ is transferred (see section 3.2) to the receiving prover k. After that, prover i integrates clauses $R_i^j \subseteq \bigcup_{k \neq i} S_{k,i}^j$ of provers k ($k \neq i$) selected by a receive-referee into its search state.

3.2 Achieving cooperation between provers

In order to realize cooperation between provers by exchanging some deduced clauses we must at first introduce techniques for identifying clauses to be exchanged and for transferring these clauses to other provers. Thus, we have to deal with the architecture of our cooperative system and with the realization of referees. Furthermore, we sketch aspects regarding the completeness of saturation-based provers when integrating clauses into their search states.

3.2.1 General Architecture

There are two main aims when selecting clauses from the active clauses of prover i which should be sent to prover j: On the one hand, it is sensible to utilize as much knowledge as possible in the selection process so as to perform an "optimal" selection regarding the systems of active clauses of i and j. On the other hand, the selection and transmission of clauses should be very efficient in order to reduce the overhead caused by the cooperation. However, these are conflicting goals: If we, e.g., use merely *local knowledge* for selecting clauses, i.e. knowledge about the system of the sender and its history, we can select and transfer clauses very efficiently. We can select clauses at the site of the sender and send these clauses via broadcast to all receivers. This method, however, allows only to utilize a minimum of knowledge (only knowledge about the sender and its preferences) and hence concrete *needs* of receiving provers are not considered. The other extreme is to select clauses according to *global knowledge* about the systems of the sender and the receiver. If we have so much knowledge it is possible to perform an optimal selection regarding i and j. But this kind of selection requires the highest (communication) costs because the complete systems of sender and receiver must be simultaneously evaluated (which has to be realized by sending all information of the provers, including their history, to all others).

Thus, we have chosen an approach that realizes a compromise between the use of local and global knowledge or success-driven and demand-driven selection. It falls back on knowledge about the sender and receiver in a rather efficient way: We employ an individual *send-referee* for each receiver of clauses at the sender site which selects clauses from the active clauses of the sender. Moreover, each receiver employs an additional *receive-referee* which filters some clauses from the set of clauses it receives from the send-referees of other provers. This receive-referee takes the current set of clauses of the receiver into account and hence its needs. Thus, on the one hand send- and receive-referee can use knowledge about the sender and the receiver of clauses, respectively. On the other hand, the selection process is rather efficient: In the case that our team consists of n provers, $n-1$ selections take place at each sender site.

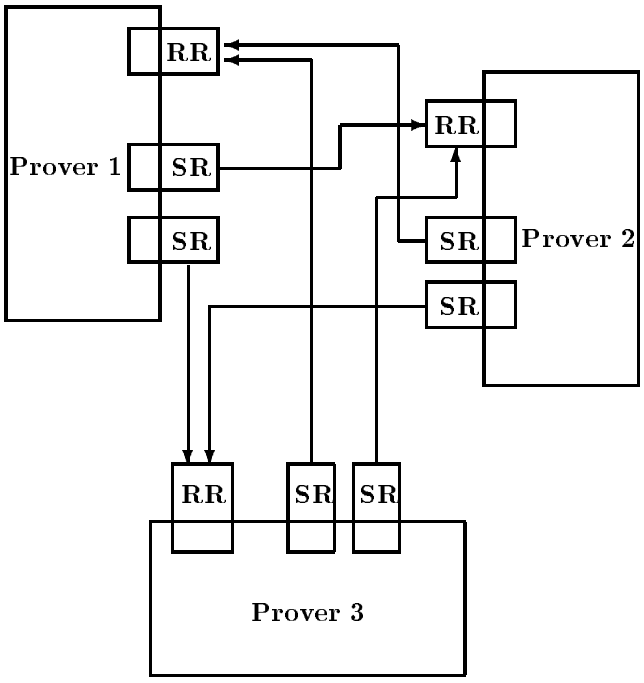

Figure 1: TECHS architecture for 3 provers

After that, $n-1$ (rather small) sets of clauses must be transferred to the receiving provers and then one additional selection out of the set of incoming clauses must be performed.

The architecture of a system based on TECHS is depicted in figure 1. The send-referees (SR) and receive-referees (RR) are displayed half inside and half outside the provers because they can be realized either as parts of the provers or as independent processes. Realizing referees as parts of the provers necessitates more implementational effort but allows to have access to internal data of the provers. Since this allows the development of more powerful referees we decided to choose this alternative for our experiments.

In order to exchange clauses between referees it is important that they employ a communication language that both send- and receive-referees understand. Consider the situation that a send-referee obtains clauses in format i and its associated receive-referee has to give some clauses to a prover which employs format j. Then, it is possible to implement receive-referees that understand many different formats of clauses and can transform them into the relevant format j. Another possibility is to employ a specific transfer language: Send-referees transfer their selected clauses from format i into this transfer language, receive-referees then transform received clauses into format j. The main disadvantage of the latter method is that two transformations are needed. However, it has the advantage that a new prover can simply be added to the cooperative system because its send- and receive-referee must only understand two clause formats. We used this second alternative and employed the syntax format of the *DFG Schwerpunkt Deduktion* ([HKW96]) for exchanging clauses.

3.2.2 Realizing referees

The quality of the referees heavily influences whether or not cooperation between the provers is successful. If referees select at least some clauses that contribute to a proof of a given problem and these clauses are not already in the data base of a receiving prover, then this prover may succeed faster. Especially, if received clauses were generated very late by the receiving prover, then a substantial speed-up of the search would be achieved.

All kinds of referees employ a *selection function* φ for the selection of clauses. φ can employ several judgment functions $\psi_1, \ldots, \psi_n$. These functions ψ_i associate a natural number with each clause C. A clause is considered the better the higher the value $\psi_i(C)$ is. φ eventually selects the clauses with the best judgments. In our experiments we selected a certain percentage p_i of clauses with each function ψ_i.

The different kinds of knowledge included in judgment functions determine how the quality of a clause is measured. On the one hand, it is possible to employ only knowledge about the sender, on the other hand also knowledge about the receiver can be used. Knowledge about the receiver can again be divided into two parts: It is possible to have knowledge about the search state of the receiver (its set of active and passive clauses). Moreover, one can have knowledge about its method to modify the search state in future (its heuristic).

3.2.3 Send-referees

A send-referee in a system based on the TECHS approach consists of a pair (S, φ) of a *filter predicate* S and a selection function φ. The prover that receives the results of the send-referee will get those clauses in the cooperation phases that pass through the filter and that are selected by φ.

The filter predicate S limits the set of clauses that are eligible for transmission to other provers. Only clauses C with $S(C)$ pass through this filter. Typically, clauses are filtered out that are redundant (with respect to the receiving provers). Redundant are all clauses that were selected in previous cooperation phases. Additionally, clauses should be filtered out that a receiver cannot use in its inference mechanism (e.g. non-unit clauses if they should be sent to an equational prover).

The following judgment functions are sensible for selecting clauses: The easiest method –that does not employ special knowledge about the receiver– is to select clauses that are quite "general". This is especially important for saturation-based provers because such clauses can often be used in order to subsume or rewrite other clauses. Moreover, clauses should not have literals with a "deep" term structure because this often prevents literals from matching others. ψ_G utilizes these criteria for judging clauses.

Definition 3.1 The judgment function ψ_G defined on clauses can be computed by $\psi_G(\{l_1, \ldots, l_n\}) = -\sum_{i=1}^{n} \psi_G(l_i)$. For literals, $\psi_G(l) = \psi_G^h(l, 0)$, if l is positive, and $\psi_G(l) = \psi_G^h(l', 0)$, if $l \equiv \neg l'$. Further,

$$\psi_G^h(l, d) = \begin{cases} 1 + d & ; l \text{ is a variable} \\ 2 + d + \sum_{i=1}^{n} \psi_G^h(t_i, d + 1) & ; l \equiv f(t_1, \ldots, t_n) \end{cases}$$

Function ψ_S uses again only knowledge about the sender, namely retrospective criteria. It defines the quality of a clause by its history during the proof attempt so far. This means that all inferences the clause was part of should be considered when developing a measure for the quality of a clause. Note that by means of such a retrospective view on the performed inferences it is possible to use a posteriori knowledge whereas typical search-guiding heuristics are only able to utilize a priori knowledge.

Definition 3.2 The judgment function ψ_S is defined as follows: Let C be a clause, let $I_1, \ldots, I_k$ be expanding, $I_{k+1}, \ldots, I_n$ be contracting inference rule types. $I_j(C)$ $(1 \leq j \leq n)$ is the number of inferences of type I_j the clause C was involved in so far. Then,

$$\psi_S(C) = -\sum_{i=1}^{k} I_i(C) + \sum_{i=k+1}^{n} I_i(C)$$

ψ_S rates contracting inferences positive, expanding inferences negative because generating too many clauses complicates the search.

In a send-referee, it is also possible to employ knowledge about the receiver in form of its heuristic.

Definition 3.3 Let A_P be the set of clauses generated by a prover P and R the receiver of the clauses to be selected. R activates clauses with heuristic $\mathcal{H}_R$. Let further be $R^{max} = \max(\{\mathcal{H}_R(C) : C \in A_P\})$ and $R^{min} = \min(\{\mathcal{H}_R(C) : C \in A_P\})$. For a clause C let R^{anc} be the maximal value $\mathcal{H}_R$ of all ancestors of C from A_P, i.e. of the $C' \in A_P$ that were needed to infer C. Then, the value of C according to the judgment function ψ_H is

$$\psi_H(C) = \begin{cases} \frac{R^{max} - \mathcal{H}_R(C)}{R^{max} - R^{min}} + \left(1 - \frac{R^{max} - R^{anc}(C)}{R^{max} - R^{min}}\right) & ; R^{max} \neq R^{min} \\ 0 & ; \text{otherwise} \end{cases}$$

This function favors clauses that have a small heuristic weight according to the heuristic of the receiver. Since the receiver favors such clauses, too, it is important to avoid the transmission of redundant information. Hence, ψ_H takes into account whether there are ancestors of a clause that have a high heuristic weight. If this is the case it is not very likely that the receiver activates such clauses by itself.

It is to be emphasized that such a criterion can only be employed by a send-referee that has access to internal data of a sending prover. It is impossible for a referee that is working at the receiver site to select clauses with this criterion: In order to employ it, it would be necessary to know not only the clauses to choose from but also the inference chains the clauses were derived with. But sending the information on both clauses and inference chains is practically impossible due to the enormous amount of communication.

More knowledge about the receiver, e.g. knowledge about its set of clauses, cannot be efficiently used by a send-referee for the same reason: The whole set of clauses of the receiver would be needed.

3.2.4 Receive-referees

The judgment functions of a receive-referee employ knowledge about the search state of the receiver. Receive-referees can only select clauses from the set of clauses they have received from the send-referees of other provers. Thus, they have to choose from a small set of clauses and can therefore employ more sophisticated (and time-consuming) criteria than a send-referee. Nevertheless, the knowledge about the system of the receiver is somewhat limited and can only facilitate the selection a little bit. (In order to go beyond the limitation, for each clause the whole future proof and the role of the clause in it has to be computed, which is not feasible.) Since the receive-referee is only able to estimate which consequences the integration of certain clauses will have we must employ heuristic criteria again. Therefore, even the selection of a receive-referee might be insufficient in some cases. In general, there are two main principles of how receive-referees can be designed, a contraction-based and an expansion-based principle.

Function ψ_U follows the first principle. Basically, it favors clauses that might often be involved in contracting inferences in future.

Definition 3.4 The judgment function ψ_U is defined as follows: Let $\mathcal{F}_R^A$ be the set of active clauses of the receiver, let C and D be clauses. Let $\delta(C, D)$ be equal to 1 if C is able to contract D, and 0 otherwise. Then,

$$\psi_U(C) = \sum_{C' \in \mathcal{F}_R^A} \delta(C, C')$$

Hence, $\psi_U(C)$ counts how often C could be used in order to contract active clauses of the receiver if it were integrated into its search state. Note that $\psi_U(C)$ can efficiently be computed because usually the set of active clauses is rather small.

Since we consider expansion inferences to be negative we require that expansion inferences involving C should at least contribute to a proof with a high probability. Function ψ_{SG} tries to estimate this in the following way.

Definition 3.5 Let ν be a function defined on pairs of clauses. Let $\mathcal{F}_R^A$ be the set of active clauses of the receiver. Let C be a clause to be judged. Then,

$$\psi_{SG}(C) = \sum_{C' \in \mathcal{F}_R^A} \nu(C, C')$$

$\psi_{SG}(C)$ sums the value $\nu(C, C')$ for each active clause C' of the receiver. $\nu(C, C')$ should be the higher the more likely C and C' contribute both to a proof and hence possibly also a descendant of C and C' may contribute to a proof. E.g., the similarity of C to a goal regarding the definitions from [DF94] could be a hint that C contributes to a proof. Exact definitions of ν depend on the concrete application domain. Hence, they can be found in sections 4 and 5.

3.2.5 Integrating clauses

The integration of clauses into the system of a saturation-based prover can simply be performed by treating the clauses as selected potential clauses. That is, the received clauses are activated in an arbitrary order. This method gives rise to the question whether or not a prover which performs fair derivations may become incomplete when it periodically integrates clauses from others. As pointed out in [Fuc97] a saturation-based prover might become incomplete if it only employs an arbitrary fair heuristic. But a stronger condition than fairness, *fairness despite disturbance*, is sufficient in order to guarantee completeness. Fortunately, most fair selection strategies are also fair despite disturbance. The main idea of fairness despite disturbance is that a heuristic must not always favor the activation of received clauses or descendants of them before the activation of one of its potential facts. For details we refer the reader to [Fuc97].

3.3 Initialization of heterogeneous teams

Now, we shall deal with the assignment of clauses from the original clause set $\mathcal{M}$ to the provers which are part of the cooperating system. We outline heuristic methods for this assignment. Moreover, we examine conditions on the assignment and on the referees that guarantee completeness of the whole system.

3.3.1 Heuristics for clause assignment

First, we consider the case that only universal provers are part of our cooperating system. Then, we set $\forall i : \gamma_i(\mathcal{M}) = \mathcal{M}$, i.e. all provers obtain the original clause set as input. It is to be emphasized that we are mainly interested in a system of cooperating provers, not in distributing or partitioning the original problem. Hence, in this case we do not partition the original clause set in contrast to distribution methods like *clause diffusion* ([BH95]).

In the case that specialized provers are part of our team we must take care of the fact that they can only deal with clauses from their relevant sub-logic. Hence, the easiest way is to use the following system of functions γ_i:

$$\gamma_i(\mathcal{M}) = \begin{cases} \mathcal{M} & ; i \text{ is a universal prover} \\ \mathcal{L}_i(\mathcal{M}) & ; i \text{ is specialized in sub-logic } L_i, \mathcal{L}_i(\mathcal{M}) = \{C : C \in \mathcal{M}, \\ & \quad C \text{ is a well-formed formula to } L_i\} \end{cases}$$

Note that this kind of assignment of initial clauses to provers often leads to the situation that specialized provers cannot solve proof problems by themselves. Nevertheless, they can efficiently work as kind of lemma generators for universal provers and can produce important consequences of the initial clause set that lie deep within the search space of a universal prover. Consider, e.g., the case that we have an equational prover as specialized prover which employs unfailing completion. If a proof problem contains positive equations and other non-unit clauses (possibly not containing equality), in general the prover is not able to solve the problem. However, it can at least produce many

logic consequences of the equations which might help a universal prover to conclude the proof.

Since specialized provers are mainly used as lemma generators a sensible refinement is that they concentrate on certain regions of the search space. Hence, it is reasonable to give a specialized prover i only a subset of $\mathcal{L}_i(\mathcal{M})$ and the prover produces only logic consequences of this subset. Possible criteria for eliminating clauses from $\mathcal{L}_i(\mathcal{M})$ are: On the one hand, it is possible to focus only on certain parts of the search space, e.g. on parts specified by using subsets of the predicate and function symbols. Hence, a specialized prover can very quickly produce logic consequences that universal provers will probably derive quite late. On the other hand, it might be sensible to insert only clauses with certain syntactic properties into the initial clause set of a specialized prover. E.g., specialized provers may only obtain large clauses which are hard to process by universal provers, but quite efficient by specialized provers because their data structures and algorithms are optimized regarding their sub-logics.

Another possible refinement is that also the input sets of universal provers are reduced. More exactly, if i is a universal and $j_1, \ldots, j_n$ are specialized provers, $\gamma_i(\mathcal{M}) = \mathcal{M} \setminus \mathcal{D}, \mathcal{D} \subset \cup_{i=1}^{n} \mathcal{L}_{j_i}(\mathcal{M})$. Hence, a universal prover can transfer parts of the search space whose exploration would lead to high costs to specialized provers. Then, with the help of the referees it can obtain "interesting" clauses from these parts of the search space. However, completeness may be lost by using this refinement (see below). One possible criterion for reducing the input of universal provers is to omit large clauses (see also 5).

3.3.2 Completeness of heterogeneous teams

Since it might be the case that provers obtain only subsets of the initial clause set we must examine the completeness of a heterogeneous team. Henceforth, we restrict ourselves to the case that a superposition-based prover cooperates with an equational prover (employing unfailing completion). This is sensible due to two different reasons: On the one hand, this special case is sufficient to show the main principles. On the other hand, we have exactly this constellation in our experiments with heterogeneous teams (see section 5).

If the universal superposition prover obtains all initial clauses and activates clauses with a heuristic which is fair despite disturbance surely the system of cooperating provers is complete.

Thus, in the following we only examine the situation that the universal prover obtains as input the set $\mathcal{M}'$, $\mathcal{M} \setminus \mathcal{M}' \subset EQ(\mathcal{M}) = \{s = t : s = t \in \mathcal{M}\}$. The equational prover obtains the set $\mathcal{M}'' \subset EQ(\mathcal{M})$. Then, in general completeness is not guaranteed. If the universal prover never obtains a certain axiom (an equation) from the specialized prover and exactly this is needed in a proof (together with a non-unit clause) none of the provers is able to find a proof. Nevertheless, there are rather weak conditions on the input sets, on the referees, and on the heuristics of the provers that guarantee completeness. We assume that the superposition-based prover performs derivations with disturbance (i.e. with periodically receiving and integrating clauses of the other prover) $\mathcal{M}' = \mathcal{M}_0 \vdash_{sup,dist} \mathcal{M}_1 \vdash_{sup,dist} \ldots$ The system of persistent clauses is $\mathcal{M}^{\infty}$.

The equational prover performs derivations with disturbance $(\emptyset, \mathcal{M}'') = (\emptyset, E_0) \vdash_{uc,dist} (R_1, E_1) \vdash_{uc,dist} \ldots$ Its persistent rules and equations are R^∞ and E^∞, respectively. Then, the following holds:

Theorem 3.1 *Let $\mathcal{M}$ be an inconsistent set of clauses containing (positive) unit equations. Let P_1 be a superposition prover, P_2 be a prover based on unfailing completion. Both provers obtain initial clause sets $\mathcal{M}' \subset \mathcal{M}$ and $\mathcal{M}'' \subset \mathcal{M}$, respectively, where $\mathcal{M} \setminus \mathcal{M}' \subset EQ(\mathcal{M})$, $\mathcal{M}'' \subset EQ(\mathcal{M})$, and $\mathcal{M} = \mathcal{M}' \cup \mathcal{M}''$. P_1 employs heuristic $\mathcal{H}_1$, P_2 heuristic $\mathcal{H}_2$. Both heuristics are fair despite disturbance. Moreover, the referee of P_2 is designed in such a way that it eventually selects all $C \in R^\infty \cup E^\infty$ and transmits these clauses to P_1. Then, there is a set $\mathcal{D}$ such that $\mathcal{M}' \vdash^*_{sup,dist} \mathcal{D}$ and $\Box \in \mathcal{D}$.*

Proof: Prover P_2 generates the systems of persistent rules and equations R^∞ and E^∞, with its inference rules and by using clauses from prover P_1, such that $=_{\mathcal{M}''} \subseteq =_{R^\infty \cup E^\infty}$. The fairness despite disturbance of the heuristic of prover P_2 guarantees that for all $s_i = t_i \in \mathcal{M}''$ holds: $s_i =_{R^\infty \cup E^\infty} t_i$ $(1 \leq i \leq n)$. Thus, for each $s_i = t_i$ there is a proof with persistent rules and equations. $B_i \subseteq R^\infty \cup E^\infty$ denotes the set of clauses needed in a proof for $s_i = t_i$. Then:

$$(*) \quad \mathcal{M}' \cup \bigcup_{1 \leq i \leq n} B_i \text{ is inconsistent} \quad \text{iff} \quad \mathcal{M}' \cup \mathcal{M}'' = \mathcal{M} \text{ is inconsistent}$$

Due to our condition regarding the referee of prover P_2 all clauses from $\cup_{1 \leq i \leq n} B_i$ will eventually be integrated into the set of clauses of prover P_1. Hence, by using $(*)$, prover P_1 has an inconsistent set of clauses after the integration of all these clauses. Because of the fact that superposition is refutationally complete and $\mathcal{H}_1$ is fair despite disturbance prover P_1 is able to derive the empty clause. $\Box$

As we can see, we must take care in the fact that all equations which are eliminated from the system of the universal prover are in the system of the specialized prover. The condition that both provers must employ a heuristic which is fair despite disturbance is not a grave restriction because this is also required from the complete prover when the provers simply exchange clauses without dividing the initial clause set. Referees with the above property can quite easily be designed, e.g. by giving time stamps to activated clauses and selecting in each cooperation phase the oldest clause that has not been selected in an earlier phase (besides other clauses selected using various criteria). Note that similar conditions are also proposed in [BH97]. However, there only the distribution of a single prover is examined. We, however, examine the coupling of heterogeneous provers in that sense that they employ different heuristics and orderings. Moreover, we are able to use provers in the network whose calculus is not complete for the logic the proof problems are specified in.

4 Homogeneous Teams via TECHS

Henceforth, we describe a case study for coupling different incarnations of the same prover, resulting in a homogeneous team. Firstly, we outline the basics of our experimental setting. Secondly, we report on some experimental results.

4.1 Experimental setting

A homogeneous team of provers consists of different incarnations of one basic prover. That is, the provers differ from each other only in the parameter settings which is mainly the heuristic they employ for activating clauses or the ordering used in some inferences.

We chose the area of full first-order theorem proving for our case study in order to demonstrate the generality of our approach. Note that we have also coupled homogeneous provers in the (specialized) area of condensed detachment with some success (see [FD97]). In our case study, we coupled different incarnations of the prover SPASS. This prover is already very powerful when working sequentially as results of theorem proving competitions reveal (see, e.g., [SS97]). In order to couple incarnations of SPASS it was necessary that SPASS could employ several heuristics. Since SPASS has merely one heuristic at its disposal we were forced to add new heuristics which are slightly modified variants of the basic heuristic. Note that –starting with obtaining SPASS via ftp– we needed one man-month for the whole process of extending SPASS to a cooperative prover. Hence, our cooperation concept proved to be easily implementable.

We coupled two different incarnations of SPASS. One incarnation was the SPASS standard heuristic, the heuristic of the other incarnation was realized by weighting function symbols which occured in axioms but not in conjecture clauses with the value 5 and the remaining function symbols with the value 2. It is to be emphasized that both heuristics are fair despite disturbance. Each prover employed a send- and a receive-referee. The send-referee of each prover selected 24 clauses, the receive referee then selected 10 clauses from these clauses. Note that the exchange of clauses was simply organized via files. The send-referees fell back on judgment functions ψ_S, ψ_G, and ψ_H as described in section 3.2. 8 clauses were selected with each function. The receive-referees employed functions ψ_U and ψ_{SG}, and used each function for identifying 5 clauses to be integrated.

In order to define ψ_{SG} we have to explain the function ν which is the basis of ψ_{SG}. Recall that $\nu(C, C')$ should estimate whether or not a descendant of C and C' is likely to contribute to a proof. Because of the fact that a receive-referee should not perform inferences by itself simple criteria have to be used for such an estimation. We assume that a clause C contributes to the refutation of another active clause C' if we can perform a superposition step with C and C'. However, a superposition step does not necessarily lead to a derivation of the empty clause. On the contrary, the clause length of the resulting clause can even increase. Hence, we assume that C only contributes to a refutation of C' if at least one descendant of C and C' exists which does not have more literals than C or C'. In order to reduce the amount of computation needed to check this we use the following heuristic: If C is a unit and has a descendant with C' we assume it to be contributing to the refutation of C'. If C is not a unit clause we consider it to be only contributing to a proof of C' if this clause is a unit and has a descendant with C. Furthermore, it is possible to take the length of the non-unit clauses into account: When performing inferences with short clauses it might be easier to derive the empty clause than when deriving new clauses with very long ones.

Definition 4.1 Let C and C' be clauses, let $r(C, C')$ be the number of clauses derivable

problem	SPASS	mod. SPASS	TECHS	OTTER
LCL196-1	292.4s	311.7s	83.8s	34.9s
LCL163-1	10.0s	11.9s	7.5s	–
GRP048-2	23.3s	17.4s	8.7s	101.9s
GRP148-1	951.2s	253.1s	184.7s	70.3s
GRP169-1	80.1s	15.7s	13.7s	9.9s
GRP169-2	56.1s	26.2s	9.7s	9.5s
GRP174-1	8.0s	8.3s	5.8s	9.6s
RNG018-6	639.9s	199.7s	152.3s	0.5s
NUM009-1	8.1s	6.3s	2.6s	21.5s

Table 1: Coupling incarnations of SPASS via TECHS

by applying superposition to C and C'. Then, if C is a unit we define the evaluation function for superposition ν^{sp} by

$$\nu^{sp}(C, C') = \begin{cases} \infty & ; \exists \sigma : \sigma = mgu(C, \neg C') \\ \frac{r(C,C')}{|C'|} & ; \text{otherwise} \end{cases}$$

Otherwise, if C is not a unit clause, we define

$$\nu^{sp}(C, C') = \begin{cases} 0 & ; C' \text{ is not a unit clause} \\ \frac{r(C,C')}{|C|} & ; \text{otherwise} \end{cases}$$

4.2 Experimental results

We conducted experimental studies in the light of problems taken from the problem library TPTP ([SSY94]). As our test set we chose problems from the CADE-13 ATP system competition ([SS97]) and selected those from the categories 'unit equality' and 'mixed'.

Table 1 presents an excerpt of our results obtained when tackling hard and medium problems (the runtime is at least 8 seconds on a SPARCstation 20). Column 1 shows the name of the problem. The remaining columns display the runtimes (obtained on one or two SPARCstations 20) of the basic heuristic of SPASS (SPASS), our newly implemented heuristic (mod. SPASS), our cooperating system of provers (TECHS), and OTTER in its auto-mode (see [McC94]). We use the runtime of OTTER as a point of reference merely in order to show that we are not dealing with trivial problems. More interesting, however, is the comparison of the single heuristics with our system of cooperating provers.

The speed-ups w.r.t. the better of the two heuristics for each problem range from 1.3 to 3.5 with more than half of the problems having a (super-linear) speed-up of 2 or more. So, despite the overhead produced by TECHS, that includes interchange of data via files, the performance of SPASS was definitely improved.

5 Heterogeneous Teams via TECHS

We also conducted a case study using different provers in a team and in the following report on achieved results.

5.1 Experimental setting

In order to couple heterogeneous provers we let a universal prover and a prover specialized in pure equational logic cooperate. This is interesting for two different reasons: Firstly, equality is involved in many problems of first-order logic and usually universal provers have serious difficulties when dealing with such problems. Hence, cooperation with a prover specialized in equational logic might be the right way for increasing performance. Secondly, there exist many high-performance provers for pure equational logic whose applicability is rather restricted because most problems contain –besides unit equations– also non-unit clauses. Thus, by employing such specialized provers in a network of cooperating provers they can contribute to the solution of problems of full first-order logic and their applicability can hence be increased.

We have again chosen the superposition-based prover SPASS with its basic heuristic as universal prover. We have chosen the unfailing completion prover DISCOUNT as specialized prover. Note that DISCOUNT is superior to SPASS in pure equational logic because it has goal-oriented heuristics ([DF94]) at its disposal which are only suitable for equational logic. Moreover, its implementation and data structures are specialized so as to very efficiently deal with equations. Hence, DISCOUNT is more powerful than a version of SPASS restricted to equational logic. In our case study, DISCOUNT activates clauses with such a goal-oriented heuristic. Note that these heuristics are not fair, i.e. also not fair despite disturbance (see [Fuc97]). However, since the universal prover SPASS employs a heuristic which is fair despite disturbance the cooperating system is nonetheless complete (if SPASS obtains all input clauses). By employing these provers we have indeed a heterogeneous system: The provers employ different calculi, different orderings, and different heuristics. Moreover, their implementation is optimized regarding different aims.

Each prover employed send- and receive-referees which selected 12 clauses at the sender site and 6 clauses at the receiver site, respectively. The referees of SPASS were designed as described in section 4.1. The send-referee of DISCOUNT selected clauses with functions ψ_G, ψ_S, and ψ_H. The receive-referee fell back on functions ψ_U and ψ_{SG}. The function ν^{uc} which is needed by ψ_{SG} is defined as follows:

Definition 5.1 Let ω be a similarity function (based on function CPinGoal as described in [DF94]) between a positive equation and a negative equation (goal). Let C, C' be unit equations. Then, we define

$$\nu^{uc}(C,C') = \begin{cases} \omega(C,C') & ; C \equiv \{s=t\}, C' \equiv \{\neg u = v\} \\ \omega(C',C) & ; C \equiv \{\neg s = t\}, C' \equiv \{u = v\} \\ 0 & ; \text{otherwise} \end{cases}$$

We examined three different variants for an initial assignment of clauses to the provers. Let $\mathcal{M}$ be the original set of input clauses, $EQ(\mathcal{M})$ be the set of positive equations in $\mathcal{M}$, and $EQ_{neg}(\mathcal{M})$ be the set of negative equations in $\mathcal{M}$. Then, the first variant (*basic assignment*) is to give SPASS all clauses from $\mathcal{M}$, and DISCOUNT all equations from $EQ(\mathcal{M})$ as axioms and all inequations from $EQ_{neg}(\mathcal{M})$ as goals. The second variant (*input reduction of* DISCOUNT) is to again give SPASS all clauses but to reduce the input of DISCOUNT to "difficult" parts. More exactly, we eliminated from the input set $EQ(\mathcal{M}) \cup EQ_{neg}(\mathcal{M})$ of DISCOUNT 10% of the positive equations with the lowest weight (the lowest number of symbols). Finally, the third variant (*input reduction of* SPASS) is not to reduce DISCOUNT's input, i.e. it obtains $EQ(\mathcal{M}) \cup EQ_{neg}(\mathcal{M})$, but to eliminate some equations from the input of SPASS. We deleted 10% of the equations having the highest weights. Note that it is sometimes the case (especially in the HEN domain) that some of these equations are redundant and that SPASS can solve the problem much faster without using them. In order to avoid this, we only eliminated such clauses that are really necessary for a proof. That is, SPASS is not able to derive the empty clause by only producing logic consequences of its reduced input set but it really needs support from DISCOUNT. It is to be emphasized that we employed these variants only when dealing with problems that are not specified in pure equational logic. In that case both theorem provers obtain all equations as input.

5.2 Experimental results

We experimented in the light of problems from the TPTP. We dealt especially with the ROB and HEN domain. We chose those problems which contain positive unit equations that can be given to DISCOUNT (we consider literals $P(t_1, \ldots, t_n)$ to be equations $P(t_1, \ldots, t_n) = true$). Sometimes (especially in the HEN domain) the situation occurred that the completion of DISCOUNT stopped and SPASS could not transmit unit equations to DISCOUNT. In order to deal with such problems we extended DISCOUNT in that case so as to let it partly work with horn clauses: We allowed horn clauses to pass through the filter of SPASS. Then, DISCOUNT tried to derive unit equations by applying hyper-resolution to its set of equations and the received horn clauses. After that, the horn clauses were deleted and the derived equations were treated as received equations.

By employing the described experimental setting we could achieve a consistent gain of efficiency in both examined domains HEN and ROB. Table 2 presents an excerpt of our experimental results in these domains, enriched with some problems taken from other domains. Column 1 again shows the name of the problem, columns 2 and 3 the sequential runtimes of the provers (again obtained on a SPARCstation 20). As one can see, SPASS sometimes outperforms DISCOUNT for unit equality problems although it is in general not as powerful as DISCOUNT in equational logic. This is because the used goal-oriented heuristics of DISCOUNT are especially well-suited when combining them with conventional heuristics but sometimes fail when working alone. Columns 4–6 display the results obtained with our three variants of TECHS. Note that for problems of unit equality we only used the first variant (basic assignment). Then, we used the same runtimes for the second and third variant as for the first variant.

problem	SPASS	DISCOUNT	TECHS			OTTER
			bas. assign.	inp. red. DISCOUNT	inp. red. SPASS	
BOO007-4*	403.4s	–	19.3s	19.3s	19.3s	10.9s
CIV001-1	24.9s	–	12.4s	8.4s	12.3s	7.4s
GRP169-1*	80.1s	–	41.9s	41.9s	41.9s	9.9s
GRP177-2*	–	–	58.3s	58.3s	58.3s	–
GRP179-1*	–	–	80.7s	80.7s	80.7s	–
HEN003-1	28.2s	–	13.3s	11.6s	12.9s	1.3s
HEN003-2	58.1s	–	21.5s	20.4s	14.5s	1.3s
HEN005-1	10.3s	–	4.8s	4.6s	5.0s	1.6s
HEN005-2	55.2s	–	34.2s	40.1s	31.7s	3.9s
HEN006-1	851.9s	–	632.8s	621.3s	627.2	7.2
HEN006-3	16.2s	–	7.0s	7.0s	14.1s	29.4s
HEN006-6	12.0s	–	4.7s	4.5s	4.1s	19.4s
HEN009-5	309.9s	–	105.9s	42.0s	35.7s	119.7s
HEN010-5	68.7s	–	14.7s	12.7s	9.8s	48.6s
HEN011-4	12.8s	–	10.2s	10.2s	9.8s	–
HEN011-5	41.2s	–	26.2s	18.8s	17.4s	–
LCL143-1	16.1s	–	1.7s	1.8s	1.7s	1.0s
LCL146-1	12.4s	–	8.7s	8.3s	2.3s	271.7s
LCL163-1*	10.0s	12.0s	7.7s	7.7s	7.7s	–
LDA010-2	–	–	682.7s	649.6s	682.7s	–
ROB005-1*	–	109.6s	39.9s	39.9s	39.9s	44.9s
ROB008-1*	–	98.8s	33.3s	33.3s	33.3s	0.4s
ROB011-1	105.3s	–	21.9s	5.7s	14.9s	0.6s
ROB014-2	95.3s	–	87.3s	90.8s	75.6s	229.2
ROB016-1	9.8s	–	4.6s	4.5s	2.6s	0.7s
ROB022-1*	15.1s	–	6.9s	6.9s	6.9s	5.0s
ROB023-1*	204.6s	–	4.4s	4.4	4.4	2.2s

Table 2: Coupling SPASS and DISCOUNT via TECHS

Such problems are marked with an asterisk. The last column presents the runtime of OTTER's auto-mode.

Table 2 shows that the cooperation of heterogeneous theorem provers leads to even better results as before. On the one hand, we have rather high speed-ups (up to the factor 46). On the other hand, the system of cooperating provers was able to solve 3 problems no single prover was able to solve when working alone (problems GRP177-2, GRP179-1, and LDA010-2). Since most of the problems are in first-order logic with equality, DISCOUNT was not able to solve them.

The reduction of the input of DISCOUNT so as to let the prover mainly work on difficult parts of the search space led in most cases to (moderate) speed-ups. However, there are problems where this input reduction could significantly improve on the basic variant of TECHS (e.g., HEN009-5, LDA010-2, ROB011-1). When reducing the input of SPASS we can observe similar results. In most cases the runtimes slightly decrease, and there are some problems where we achieve quite a high gain of efficiency (e.g., HEN009-5, LCL146-1, HEN011-5). Note that the results become worse when reducing the input of

both SPASS and DISCOUNT. Perhaps the criteria of our referees are too vague in order to cope with this situation.

6 Discussion and Future Work

We have presented the TECHS methodology for realizing cooperation between heterogeneous theorem provers. The main principles of TECHS are to let several provers work in parallel and to achieve cooperation by periodically exchanging clauses. The clauses are selected by send- and receive-referees. In the case that specialized provers are in the network of cooperating provers, TECHS allows for a division of the original input so as to efficiently use specialized provers as lemma generators.

We examined the methodology theoretically and discovered weak conditions on the heuristics that guarantee completeness even when a prover periodically receives and processes clauses from other provers. In the case that only some clauses of the original input are assigned to different provers, we pointed out weak conditions on this assignment, on the referees, and on the heuristics of the provers that are sufficient for completeness. Besides theoretic studies we developed and described heuristics for referees and the initial clause assignment that proved to be successful in two case studies. In addition, TECHS can be easily integrated into existing high-performance provers.

There already exist some approaches for distributed theorem proving. Hence, we want to compare TECHS with related approaches. The referee concept of TECHS is essentially adapted from the TEAMWORK method ([Den95]). However, the main difference between TEAMWORK and TECHS is that the latter approach is intended for coupling heterogeneous provers, the first for homogeneous provers. TEAMWORK requires that periodically different provers are judged and that the system of the "best" prover is the starting system of the other provers. Hence, the provers in the network must be able to work on the same kinds of system of clauses. As a matter of fact, TEAMWORK is only sensible if different incarnations of one prover should be coupled. In addition, TEAMWORK is not able to use demand-driven criteria for selecting results to be interchanged.

The clause diffusion method ([BH95]) is essentially a distribution approach of *one* basic prover. It requires an initial partition of the original input and distributes the clauses to different processors. On each of these processors the same prover works on these clauses. Then, in order to remain complete generated clauses must be sent around in order to perform inferences involving them. Hence, clause diffusion does not allow for the use of referees to reduce the amount of communication and computation. Moreover, in its current form it is—as already mentioned—not suitable for heterogeneous provers.

Heterogeneous teams were—to our knowledge—so far only used in [Sut92]. There, provers employing different calculi worked in parallel and asynchronously exchanged each generated clause. Again, no referee concept was used and hence the results were not very convincing.

It should be noted that all known distribution approaches for theorem provers, with the exception of TECHS require rather massive changes to provers in order to allow

them to use the approaches. They also do not allow for the cooperation of different existing provers.

Future work should deal with increasing the heterogeneity of the prover network. In particular, we are interested in also integrating analytic tableau-style provers into our network which already lies within the TECHS framework.

References

[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.

[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.

[BH95] M.P. Bonacina and J. Hsiang. The Clause-Diffusion methodology for distributed deduction. *Fundamenta Informaticae*, 24:177–207, 1995.

[BH97] M.P. Bonacina and J. Hsiang. On the representation of dynamic search spaces in theorem proving. In *Proc. of the Int. Computer Symposium*, pages 85–94, 1997.

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

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

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

[Fuc97] D. Fuchs. Coupling Generating Theorem Provers by Exchanging Positive/Negative Information. Technical Report SR-97-07, University of Kaiserslautern, Kaiserslautern, 1997.

[HKW96] Reiner Hähnle, Manfred Kerber, and Christoph Weidenbach. Common Syntax of DFG-Schwerpunktprogramm "Deduktion". Technical report 10/96, Universität Karlsruhe, Fakultät für Informatik, 1996.

[HR87] J. Hsiang and M. Rusinowitch. On word problems in equational theories. In *Proc. ICALP87*, pages 54–71. LNCS 267, 1987.

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

[SS97] G. Sutcliffe and C.B. Suttner. The results of the cade-13 ATP system competition. *Journal of Automated Reasoning*, 18(2):271–286, 1997.

[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.

[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.