Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

Distributing equational theorem proving

J. Avenhaus, J. Denzinger

SEKI Report SR-93-06

Distributing equational theorem proving[1]

J. Avenhaus · J. Denzinger
Fachbereich Informatik, Universität Kaiserslautern
6750 Kaiserslautern

{avenhaus . denzinge}@informatik.uni-kl.de

Abstract

In this paper we show that distributing the theorem proving task to several experts is a promising idea. We describe the team work method which allows the experts to compete for a while and then to cooperate. In the cooperation phase the best results derived in the competition phase are collected and the less important results are forgotten. We describe some useful experts and explain in detail how they work together. We establish fairness criteria and so prove the distributed system to be both, complete and correct. We have implemented our system and show by non-trivial examples that drastical time speed-ups are possible for a cooperating team of experts compared to the time needed by the best expert in the team.

[1] A short version of this paper appears in the proceedings of RTA-93.

1 Introduction

The success of general theorem provers is limited by the fact that even for relatively simple problems the search space for finding a proof becomes too large. There are several possibilities to deal with this problem. On the syntactical level one may restrict the search space by designing powerful inference rules or by imposing order restrictions. On the semantical level one may incorporate domain specific knowledge and proof plans. On the machine level one may use parallelism. In this paper we propose the team work method (see [De93]). it easily allows one to combine these ideas.

The basic idea of the team work method is as follows: There is a supervisor that activates severals experts – running on different processors – to work on the given problem. Each expert is a prover by itself, it may focus on a subproblem (for example on a special part of the database) and follow its own heuristics. So for a given amount of time the experts work independently and compete. When this time is elapsed the supervisor stops the experts and calls the referees to judge the work done by the experts. Based on the referee reports one of the experts is declared to be the winner. The referees determine the best results of the losers and send these results to the winner. Now the supervisor creates a new team of experts and starts a new round. The whole process stops as soon as one expert has found a proof. This approach seems to be very flexible: If the supervisor has some knowledge on the problem he can decompose the problem into subproblems, select domain specific experts and combine their results. This allows one to execute a given proof plan efficiently. On the other side, if only little is known about the given problem, the supervisor may start a standard team of experts and after each round exchange those experts by others that did not contribute to solving the problem. In this case it is indispensable to forget those results that will not help to find the proof. Otherwise the database would explode and the "wrong" experts would prevent the system to find a proof. It is one task of referees to extract the useful results derived by the experts.

It is the aim of this paper to make these ideas precise and to study the problems that come along with the approach. We believe that the approach is useful for a wide class of theorem provers based on inference rules that generate new and simplify old facts (e.g. resolution based provers), but we restrict in this paper to pure equational reasoning. To be precise, the underlying prover is the unfailing Knuth-Bendix completion procedure [BDP89]. In this case we are able to present different useful experts. We analyze the tasks of the supervisor and the referees and we discuss communication problems. We give simple criteria to guarantee fairness, they imply that the whole distributed system is both, correct and complete. Experiments show that the approach is promising. For example, we prove the ring example of Stickel [St84] (a ring satisfying $x^3 = x$ is commutative) without AC-unification and AC-rewriting with a team consisting of two experts in 308 seconds. Here the best expert alone needs 5153 seconds. This speed-up factor much greater than two for a team of two experts is not unusual as other examples show.

The paper is organized as follows: In section 2 we review the proof method unfailing completion. In section 3 we describe the team work method in detail and we give

conditions to guarantee fairness in section 4. We discuss several experts and the tasks of the referees and the supervisor in section 5. In section 6 we discuss some examples and prove that remarkable speed-ups are possible for team experts working together. Finally, in section 7 we relate our approach to some of those known in the literature.

2 Unfailing completion as the basic proof procedure

We apply the team work method outlined in the introduction to purely equational reasoning. So we are interested in the following problem:

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

Let $Th(E)$ denote the set of equations over sig that hold in every model of E. By Birkhoff's theorem we have $s = t \in Th(E)$ iff s can be transformed into t by *replacing equals by equals*. It is well-known that provers based on rewriting and completion techniques developed by Knuth and Bendix [KB70] are efficient for this problem. In order to avoid abortion of the completion procedure due to the fact that equations may not be orientable into rules we use the *unfailing completion procedure* of Bachmair, Dershowitz and Plaisted [BDP89] as our basic proof procedure.

We assume the reader to be familiar with rewriting and completion techniques. For an overview see [AM90] and [DJ90]. We use the standard notations.

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

A *rule* is an oriented equation, written $l \rightarrow r$ such that $Var(r) \subseteq Var(l)$. A set R of rules is *compatible* with $\succ$ if $l \succ r$ for every $l \rightarrow r$ in R. If E is a set of equations then $R_E = \{\sigma(u) \rightarrow \sigma(v) \mid u \doteq v \text{ in } E, \sigma \text{ a substitution}, \sigma(u) \succ \sigma(v)\}$ is the set of orientable instances of equations in E. (We use $u \doteq v$ to denote $u = v$ or $v = u$.) Finally, we have $R(E) = R \cup R_E$.

Let $u \doteq v$ and $s \doteq t$ be equations in $E \cup R$. Let u/p be a non-variable subterm of u that is unifiable with s, say with most general unifier $\sigma = mgu(u/p, s)$. Then $\sigma(u[t]_p) = \sigma(v)$ is in $Th(R \cup E)$. If $\sigma(t) \not\succeq \sigma(s)$ and $\sigma(v) \not\succeq \sigma(u)$ then $\sigma(u[t]_p) = \sigma(v)$ is a critical pair of R, E. We denote by $CP(R, E)$ the set of all critical pairs of R, E. We are now ready to define the unfailing completion procedure. It works on triples of the form (E, R, g) and is parameterized by a ground reduction ordering $\succ$. Here E is a set of equations (originally the input), R a set of rules compatible with $\succ$ (originally

empty) and g a ground equation over $F \cup K$ (originally the skolemized input goal $s = t$). The completion procedure is given by a set of inference rules and a set of fairness conditions that restrict the application of the inference rules.

Definition 2.1 (Inference system $\mathcal{U}$, see [BDP89])
Let $\succ$ be a ground reduction ordering. The inference system $\mathcal{U}$ consists of the following inference rules.

(U1) *Orient an equation*
$(E \cup \{s \doteq t\}, R, g) \vdash_{\mathcal{U}} (E, R \cup \{s \to t\}, g)$ *if* $s \succ t$

(U2) *Deduce an equation*
$(E, R, g) \vdash_{\mathcal{U}} (E \cup \{s = t\}, R, g)$ *if* $s = t \in CP(R, E)$

(S1) *Delete an equation*
$(E \cup \{s = t\}, R, g) \vdash_{\mathcal{U}} (E, R, g)$ *if* $s \equiv t$

(S2) *Simplify an equation*
$(E \cup \{s \doteq t\}, R, g) \vdash_{\mathcal{U}} (E \cup \{u = t\}, R, g)$ *if* $s \longrightarrow_{R(E)} u$

(S3) *Subsume an equation*
$(E \cup \{s \doteq t, u \doteq v\}, R, g) \vdash_{\mathcal{U}} (E \cup \{s = t\}, R, g)$ *if* $u/p \equiv \sigma(s), v \equiv u[\sigma(t)]_p$ *for some* σ *and position* p *and* $u \rhd s$

(S4) *Simplify a rule, right*
$(E, R \cup \{s \to t\}, g) \vdash_{\mathcal{U}} (E, R \cup \{s \to u\}, g)$ *if* $t \longrightarrow_{R(E)} u$

(S5) *Simplify a rule, left*
$(E, R \cup \{s \to t\}, g) \vdash_{\mathcal{U}} (E \cup \{s = u\}, R, g)$ *if* $s \longrightarrow_{R(E)} u$ *using* $l \to r$ *and* $s \rhd l$

(G1) *Simplify the goal*
$(E, R, s = t) \vdash_{\mathcal{U}} (E, R, u = t)$ *if* $s \longrightarrow_{R(E)} u$

(G2) *Success*
$(E, R, s = t) \vdash_{\mathcal{U}} SUCCESS$ *if* $s \equiv t$

In this definition $\rhd$ denotes the encompassment ordering. It is the strict part of the quasi-ordering defined by $s \trianglerighteq t$ iff $\sigma(t) \equiv s/p$ for some substitution σ and some position p. Notice that we have added subsumption rule (S3) that is missing in [BDP89]. This rule is indispensable for efficiency reasons. For instance, if commutative and associative operators are present it prevents an explosion of the set E.

Using the orderings $\succ$ and $\rhd$ a proof ordering $>_p$ can be constructed such that the following holds (see [BDP89]):
If $(E, R, g) \vdash_{\mathcal{U}} (E', R', g')$ and B is a proof for $s = t$ in (E, R) then there is a proof B' for $s = t$ in (E', R') with $B \geq_p B'$. In particular, if B is a peak $s_{R(E)} \leftarrow u \rightarrow_{R(E)} t$ and $s = t$ is in E' then $B >_p B_{s,t}$ where $B_{s,t}$ is the one step proof consisting of applying the equation $s = t$.

Definition 2.2 (Fairness of a derivation sequence)
A $\mathcal{U}$-derivation is a sequence $(E_i, R_i, g_i)_{i \geq 0}$ with $(E_i, R_i, g_i) \vdash_{\mathcal{U}} (E_{i+1}, R_{i+1}, g_{i+1})$ for all i. It defines the sets R^∞ and E^∞ of persistent rules and equations by

$$R^\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} R_i \qquad E^\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} E_i.$$

The derivation is fair if either it ends with SUCCESS or else for every critical pair $u = v$ in $CP(R^\infty, E^\infty)$, there is an $i \geq 0$ and a proof B for $u = v$ in (E_i, R_i) with $B_{u,v} \geq_p B$.

It is obvious that one way to get such a proof B for a critical pair is to add the equation $u = v$ to an E_i.

The main theorem on unfailing completion now is ([BDP89])

Theorem 2.1 *Let $(E_i, R_i, g_i)_{i \geq 0}$ be a fair $\mathcal{U}$-derivation with $(E_0, R_0, g_0) = (E, \emptyset, \overline{s} = \overline{t})$ where $\overline{s} = \overline{t}$ is the skolemized version of $s = t$. We have $s = t \in Th(E)$ iff the derivation is finite and ends with SUCCESS.*

This theorem directly gives raise to a theorem prover for the problem "$s = t \in Th(E)$?" that is both, correct and complete.

Definition 2.3 (Basic prover)
A basic prover is any algorithm that with input $(E, s = t, \succ)$ produces only $\mathcal{U}$-derivations. The basic prover is fair, if it produces only fair $\mathcal{U}$-derivations.

Theorem 2.2 *Every fair basic prover started with input $(E, s = t, \succ)$ will stop and generate SUCCESS whenever $s =_E t$ holds.*

Proof: The theorem follows immediately from Theorem 2.1 and the Definition 2.3.

3 Team work completion

The team work method was mainly designed to use distributed computation in situations where almost nothing is known of how to find a proof for the problem instance $(E, s = t)$. In this case the supervisor activates a team of probably good experts (basic provers) and lets them try to solve the problem independently. He hopes that at least one of the experts is well suited for the problem instance and some of the other experts deliver valuable subresults at the right time. So after a while he stops the competition phase and starts a team meeting for cooperation. Now the work of the experts has to be judged and this is the task of the referees. So the supervisor really selects a team of expert/referee pairs. Each referee gives a report on the overall behavior of his expert and selects the most important results. On the basis of this information the supervisor declares one of the experts as the winner and the selected results of the losers are sent to the winner. Using this extended database of the winner the supervisor now starts a new round of competition and cooperation. He stops all computations as soon as one proof has been found.

So the computation time is split into rounds. The $k-th$ round has the following form:

Cooperation: The supervisor accepts the referee reports from round $k-1$. Based on this information he determines the winner and accepts the selected results of the losers. Then he selects a new n-tuple of expert/referee pairs.

Competition: The experts work independently.

Judgement: The referees prepare their reports.

This concept sounds simple. In section 6 we will demonstrate by examples that it works and that it makes remarkable speed-ups possible. Even more, the concept seems to be very flexible. Very different sorts of knowledge can be implemented either in the supervisor (e.g. proof plans) or in the experts (e.g. domain knowledge).

Clearly, the concept can only work if

a) the tasks of the supervisor and the referees are carefully examined

b) useful experts are created

c) reasonable criteria for referees to judge the work of experts are developed

d) communication time is reduced to a minimum.

We will discuss these problems in section 5. Here we describe in more detail the general form of an expert and how the cooperation of the experts is organized by the supervisor. This will allow us to develop fairness criteria for the distributed system. This discussion is on a conceptual level to simplify proofs. For implementational aspects see section 5.

By definition a basic prover is any algorithm that produces on input $(E, s = t)$ only $\mathcal{U}$-derivations. We now present the general form of a basic prover used in our system. For efficiency reasons a basic prover will apply the simplification rules from the inference system $\mathcal{U}$ with highest priority. Then for a fixed input $(E, s = t)$ its performance mainly depends on the way the rules/equations are selected to compute critical pairs by rule (U2). The experts to be described later mainly differ in their heuristics for this choice. They try to generate many critical pairs early in order to give their heuristic a chance to find a good one.

A basic prover P works on a quadruple (R, E, g, CP). Here R and E are the current sets of rules and equations, g is the current goal $\overline{s} = \overline{t}$ and CP is the set of critical pairs not processed so far. Furthermore, P has a reduction ordering $\succ$ and a function called *choose-CP* as input. P performs a while loop with the following loop invariant: R is compatible with $\succ$ and for any equation $u = v$ in E the terms u, v are incomparable by $\succ$. All critical pairs in $CP(R, E)$ are already computed and stored in CP. The function *choose-CP* selects the next element from CP to be processed. So a quadruple (R, E, g, CP) of a basic prover corresponds to the triple (E, R, g) of the inference system $\mathcal{U}$. The CP-component in the quadruple is used to keep track of the critical pairs not processed so far and for choosing a good one (according to the heuristic used) to be processed next. In more detail P has the following form (by $nf_{R(E)}(t)$ we denote

the computation of a normal form of t with respect to $R(E)$):

Procedure basic-prover

input: $(R, E, s = t, CP, \succ,$ *choose-CP*$)$
output: YES or NO or (R', E', g', CP')
begin
  while $CP \neq \{\}$ **do**
   $(l_2, r_2) :=$ *choose-CP(CP)*;
   $CP := CP \backslash \{(l_2, r_2)\}$;
   $l_1 := nf_{R(E)}(l_2)$;
   $r_1 := nf_{R(E)}(r_2)$;
   if $l_1 \neq r_1$ **then**
     if l_1 and r_1 are comparable with $\succ$ (let $l := max\{l_1, r_1\}$; $r := min\{l_1, r_1\}$)
       then $R := R \cup \{l \rightarrow r\}$;
            $CP := CP \cup \{(nf_{R(E)}(u),\ nf_{R(E)}(v)) \mid u = v$ is a critical pair
                 between R and $l = r$ or E and $l = r\}$;
            interreduce R and E;
       else $E := E \cup \{l = r\}$;
            $CP := CP \cup \{(nf_{R(E)}(u),\ nf_{R(E)}(v)) \mid u = v$ is a critical pair
                 between R and $l = r$ or E and $l = r$ or $E\}$;
            interreduce R and E;
   if $nf_{R(E)}(s) \equiv nf_{R(E)}(t)$ **then**
     answer-to-supervisor "YES";
   if interrupt-by-supervisor **then**
     answer-to-referee $(R, E, nf_{R(E)}(s) = nf_{R(E)}(t), CP)$ + statistical information;
  endwhile;
  answer-to-supervisor "NO";
end

It is easily seen that P is indeed a basic prover according to Definition 2.3. Fairness of P can now be achieved by guaranteeing that the *choose-CP* function rejects no equation in CP infinitely often.

The supervisor may not interrupt a basic prover in the middle of the while loop. An interrupt may only occur at the position indicated in the procedure *basic-prover*.

Note that simplification, also backward simplification, is a fundamental part of our basic provers and so of the distributed system also. So we do not have the bottleneck backward subsumption as, for example, the approach of Slaney and Lusk [SL90] has.

The cooperation during a team meeting is organized as follows:

(1) The supervisor determines the winner of the latest round.

(2) He accepts the selected rules/equations from the losers and integrates them into the quadruple (R, E, g, CP) of the winner by processing them as indicated in the while loop in the procedure basic-prover.

3) The supervisor determines an n-tuple of new expert/referee pairs for the new round, including the winner. He starts the $n-1$ experts (besides the winner) with the quadruple $(\emptyset, \emptyset, g, R \cup E \cup CP)$, where (R, E, g, CP) is the updated quadruple of the winner.

4 Fairness

The computation in the distributed system with input $(E, s = t)$ is controlled by a team strategy S. A *team strategy* determines in a team meeting from the referee reports the winner of the latest round and the n-tuple of expert/referee pairs for the next round. A team strategy is *complete* if for any input $(E, s = t)$ with $s = t \in Th(E)$ the result YES is produced. (By construction, if YES is produced then $s = t \in Th(E)$ holds. So every team strategy S is correct.) We are going to develop criteria for the completeness of a team strategy.

To do so we first extend the inference system $\mathcal{U}$ for describing sequential provers to an inference system $\mathcal{DU}$ for describing our distributed prover. Then we express completeness criteria for a team strategy by fairness criteria in $\mathcal{DU}$.

We extend $\mathcal{U}$ to $\mathcal{DU}$ by adding rules for describing the integration of the selected rules/equations of the losers into the database of the winner.

Definition 4.1 (Inference system $\mathcal{DU}$)
Given the ground reduction ordering $\succ$ the inference system $\mathcal{DU}$ consists of the inference rules in $\mathcal{U}$ and the two rules

(D1)	*Introduce rule*	$(E, R, g) \vdash (E, R \cup \{l \to r\}, g)$ *if* $l =_{E \cup R} r$ *and* $l \succ r$
(D2)	*Introduce equation*	$(E, R, g) \vdash (E \cup \{u = v\}, R, g)$ *if* $u =_{E \cup R} v$ *and* u, v *are* $\succ$*-incomparable.*

Lemma 4.1 *Suppose the distributed system is started with input quadruple $(\emptyset, \emptyset, g, E)$ and in every round the winner uses a given reduction ordering $\succ$. Let (R_i, E_i, g_i, CP_i) be the actual quadruple of an active winner. Then we have $(E, \emptyset, g) \vdash^*_{\mathcal{DU}} (E_i, R_i, g_i)$.*

Proof: Let $(E_{i,j_i}, R_{i,j_i}, g_{i,j_i})$ denote the system obtained by the winner of the i-th round after the judgement phase and (E_i, R_i, g_i) the system of this winner after the cooperation phase. Then we have the following derivation $(E, \emptyset, g) \vdash^+_{\mathcal{DU}} (E_{1,j_1}, R_{1,j_1}, g_{1,j_1}) \vdash^* (E_1, R_1, g_1) \vdash^+_{\mathcal{DU}} \ldots \vdash^+_{\mathcal{DU}} (E_{i,j_i}, R_{i,j_i}, g_{i,j_i}) \vdash^* (E_i, R_i, g_i) \vdash^+_{\mathcal{DU}} \ldots$. It is clear that every expert during the competition phase only uses the inference rules of $\mathcal{U}$ and therefore the inference rules of $\mathcal{DU}$. It remains to show, that the integration of the results of the losers, here denoted by $\vdash^*$, is done using inference rules in $\mathcal{DU}$.

If a loser uses the same reduction ordering $\succ$ as the winner, we can add its selected rules according to D1 and its selected equations according to D2. If a loser uses another ordering both, its selected rules and its selected equations, are added using D2. Therefore we have $(E, \emptyset, g) \vdash^+_{\mathcal{DU}} (E_{1,j_1}, R_{1,j_1}, g_{1,j_1}) \vdash^*_{\mathcal{DU}} (E_1, R_1, g_1) \vdash^+_{\mathcal{DU}} \ldots \vdash^+_{\mathcal{DU}} (E_{i,j_i}, R_{i,j_i}, g_{i,j_i}) \vdash^*_{\mathcal{DU}} (E_i, R_i, g_i) \vdash^+_{\mathcal{DU}} \ldots$ which completes the proof. □

Lemma 4.1 indicates that the distributed computation can be described as a sequential computation according to the inference system $\mathcal{DU}$. The definition of fairness of a $\mathcal{DU}$-derivation is as in Definition 2.2. Now Theorem 2.1 can be carried over.

Theorem 4.1 *Let $(E_i, R_i, g_i)_{i\geq 0}$ be a fair $\mathcal{DU}$-derivation with $(E_0, R_0, g_0) = (E, \emptyset, \overline{s} = \overline{t})$. We have $s = t \in Th(E)$ iff the derivation is finite and ends with SUCCESS.*

Proof: The proof is identical to the proof of Theorem 2.1 and can be found in [BDP89] or [De93]. □

Now we have to find fairness criteria for a team strategy $\mathcal{S}$ such that using $\mathcal{S}$ will lead to fair $\mathcal{DU}$-derivations. For an input $(E, s = t)$ the team strategy $\mathcal{S}$ may determine the basic prover P_0 as the winner several times, say for the rounds $i_0, i_1, i_2, \ldots$. Let $(R'_j, E'_j, g'_j, CP'_j)$ be the starting quadruple of P_0 in round i_j. We call $(R'_j, E'_j, g'_j, CP'_j)_{j\geq 0}$ the P_0-sequence for $\mathcal{S}$ and $(E, s = t)$. The sequence $(E'_j, R'_j, g'_j)_{j\geq 0}$ is a subsequence of the $\mathcal{DU}$-derivation $(E_i, R_i, g_i)_{i\geq 0}$ defined by $\mathcal{S}$ and $(E, s = t)$. Note that the sets E^∞ and R^∞ are always defined by $(E_i, R_i, g_i)_{i\geq 0}$. This leads us to the following fairness criteria that also weakens the restriction on the reduction ordering used by the winners.

Definition 4.2 (Fairness of a team strategy)
A team strategy $\mathcal{S}$ is fair if (1) there is a reduction ordering $\succ$, such that for the reduction ordering $\succ_i$ of the winner of the i-th round $\succ_i \subseteq \succ_{i+1} \subseteq \succ$ holds and (2) either the computation stops or there is a basic prover P_0 with an infinite P_0-sequence $(R'_j, E'_j, g'_j, CP'_j)_{j\geq 0}$ for $\mathcal{S}$ and $(E, s = t)$ such that for every critical pair $u = v \in CP(R^\infty, E^\infty)$ there is a j such that in (E'_j, R'_j, g'_j) there is a proof B for $u = v$ with $B_{u,v} \geq_p B$.

Theorem 4.2 *Every fair team strategy is complete.*

Proof: We have to show that a fair team strategy $\mathcal{S}$ leads to a fair $\mathcal{DU}$-derivation. By Theorem 4.1 we then have the completeness of the team strategy.

Condition (1) of the definition of a fair team strategy guarantees that $\mathcal{S}$ defines a $\mathcal{DU}$-derivation (see Lemma 4.1).

Condition (2) is in fact a stronger condition than fairness of a derivation, because we have to guarantee that at certain steps of the derivation, i.e. the team meetings in which P_0 is determined as the winner, we find smaller proofs for critical pairs and not after some arbitrary step of the derivation. Therefore, the fairness of any derivation produced by $\mathcal{S}$ is trivial. □

According to Theorem 4.2 a team strategy $\mathcal{S}$ is complete if for every input $(E, s = t)$ either the computation stops or an expert P_0 becomes the winner infinitely often and for P_0 conditions (1) and (2) of Definition 4.2 hold.
Note that fairness of P_0 alone is not sufficient for condition (2). It is possible that the integration of the results of the losers leads always to critical pairs that are better rated by the choose-CP function of P_0 than already existing ones. Then these already existing

equations will eventually never be selected thus leading to a contradiction to condition (2). The next definition shows us conditions for P_0 that satisfies the condition (2) of Definition 4.2. Here we identify equations that are equal up to a variable renaming.

Definition 4.3 (strongly fair)
An expert P is strongly fair if there is a quasi-ordering $\leq$ on the equations such that

- *$\{e' \mid e' \leq e\}$ is finite for every equation e*
- *choose-CP(E) is a $\leq$-minimal element in E for every set E of equations*

Lemma 4.2 *Let $\mathcal{S}$ be a team strategy, and $(E, s = t)$ an input. If expert P is strongly fair and appears infinitely often in the sequence of winners for $\mathcal{S}$ and $(E, s = t)$ then condition (2) of Definition 4.2 holds.*

Proof: We have to show that expert P has an infinite P-sequence $(R'_j, E'_j, q'_j, CP'_j)_{j\geq 0}$ for $\mathcal{S}$ and $(E, s = t)$ such that for every critical pair $u = v \in CP(R^\infty, E^\infty)$, there is a i such that in (E'_i, R'_i, g'_i) there is a proof B for $u = v$ with $B_{u,v} \geq_p B$. Remember, the sequence (E'_i, R'_i, g'_i) is a subsequence of the $\mathcal{DU}$-derivation produced by $\mathcal{S}$ and $(E, s = t)$.

Let $u = v$ be a critical pair in $CP(R^\infty, E^\infty)$. Then for an i and all $j \geq i$ we have $u \in CP(R'_j, E'_j)$. This is true, because the rules or equations that build $u = v$ are persistent and P appears infinitely often in the sequence of winners. If there is a proof B in (E'_i, R'_i, g'_i) with $B_{u,v} \geq_p B$ then we are done. This is the case, when at the time $u = v$ can be built the normal forms of u and v are identifical. Else $u = v$ has been built and put in the set CP of critical pairs (see algorithm basic prover). If $u = v$ does not appear in CP'_i then it was reduced or put in an E_k or R_k ($k < i$), and this results in a proof B for $u = v$ that is smaller or (when put in an E_k of the $\mathcal{DU}$-sequence) equal to $B_{u,v}$. Because these steps were performed before the derivation reached (E'_i, R'_i, g'_i) we know (Theorem 4.1) that there is a proof B' in (E'_i, R'_i, g'_i) for $u = v$ with $B \geq_p B'$.

Finally, if $u = v$ is in CP'_i, then there are only a limited number n of equations $u' = v'$ that are smaller than $u = v$ with respect to an ordering $\leq$ (P is strongly fair). These equations do not have to appear as critical pairs, but we know that at least after $n+1$ critical pairs the critical pair $u = v$ will be selected by P. Note that each time P being the winner at least one critical pair will be selected. Therefore there will be a $k \geq i$ and a proof B for $u = v$ in (E'_k, R'_i, g'_k) with $B_{u,v} \geq_p B$. □

Corollary 4.1 *Let $\mathcal{S}$ be a team strategy such that for every input $(E, s = t)$ the sequence of winners is either finite or it contains a strongly fair expert infinitely often. Then $\mathcal{S}$ is complete.*

It is easy to construct a strongly fair expert. For example, the experts ADD-WEIGHT and MAX-WEIGHT discussed in the next section are strongly fair. To guarantee

fairness of the team strategy, such an expert should periodically become the winner. In the meantime unfair experts may become the winner.

We can relax the condition "strongly fair" a little bit. What we really need is that the *choose-CP* function for the distinguished strongly fair expert P never rejects an equation in the CP-component infinitely often, even if not all the equations in the CP-component are generated by P itself but may be added from outside during a team meeting. There are several possibilities to guarantee this. One is indicated in Definition 4.4. Another one would be to use time stamps and let the *choose-CP* function always select the oldest equation.

Condition (1) of Definition 4.2 restricts only the reduction ordering of the winners. All the other experts in the team may use an arbitrary reduction ordering. So completeness of a team strategy is easy to achieve. There are also ways to weaken condition (1). For further details see [De93].

Note that our way of proving the team work completion to be complete can easily be adapted to prove the completeness of the team work method for other theorem proving methods, based on generation and simplification of facts.

5 Experts, referees, the supervisor and implementation aspects

5.1 Experts

Every expert is a basic prover P, its behavior is mainly determined by its *choose-CP* function. In this function the heuristic of P for traversing the search space is encoded. We have implemented generic experts according to the following classification

- using syntactic arguments
- focusing on subproblems by focusing on a subset of function symbols
- focusing on special aspects of the (completion) method
- focusing on goal-oriented deduction

We discuss some of them.

Syntactic arguments: Experiments show that it is often advantageous to process short critical pairs first (see [Hu80]). Generalizing this idea we define a numerical weight for each term. This leads to two very useful experts called ADD-WEIGHT and MAX-WEIGHT. They give precedence to those critical pairs that have a small sum (a small maximum) of the two terms in the pair. It turns out that these experts in general perform very differently. These experts can be created without any knowledge of the problem instance, so they can be used as a member of the standard team.

Focusing on function symbols: The expert POLYNOM-WEIGHT associates to every function symbol a polynomial and a constant to all variables and so it defines

a weight for each term. To focus on the operators in $F_0 \subseteq F$, one associates small polynomials to the $f \in F_0$ and large polynomials to the $f \in F - F_0$. Experience shows that this method allows a fine tuning of the search for a proof.

Focusing on the method: Sometimes it is known that a result of a subproblem is needed for the rest of the proof. In order to get that result early an unfair expert may be needed. We have implemented FORCED-DIV and PREFER-RULE. The first of these experts concentrates on a subset of the database even if there is a high risk of divergence (i.e. generating an infinite regular set of equations). The second expert only selects critical pairs that are orientable by its reduction ordering. We discuss the use of these experts more deeply in section 6 in combination with the examples *div* and *ring*.

Focusing on the goal: Experience shows that near the end of the proof often all needed results are already deduced but the prover can not find the final steps of the proof at this moment. To solve this problem we have created the expert GOAL-SIM. This expert defines a measure for the similarity between the goal and a critical pair. We have implemented several measures, they depend on the facts whether subterms of the goal and the whole critical pair or subterms of the pair and the whole goal are unifiable. This expert has proven to be very useful in the situation lined out above. It is comparable with the terminator in resolution based theorem provers using connection graphs (see e.g. [AO83]).

There is a wide variety to define other experts that use special knowledge to focus on parts of critical pairs. It seems also possible to learn heuristics from analogous successful proofs. The team work method provides a good basis to activate such an expert even if the risk is high that it will be unsuccessful. In this case its results are just forgotten – provided the situation is correctly analyzed by the corresponding referee.

5.2 Referees

A referee has to judge the work of his expert: He has to determine the appropriateness of his expert to the given situation and he has to extract the best results derived by his expert. Without special information on the given problem instance this seems to be hard and much work is to be done in this direction. Up to now we have experimented with referees that base their judgement on statistical information.

To determine the appropriateness of an expert to the given situation the referee computes a weighted sum of the following components:

- the number of rules, equations and critical pairs generated during the latest round
- the number of reductions of the goal
- the number of reductions of rules, equations, critical pairs
- the average weight of all processed critical pairs in the latest round in relation to the last k critical pairs

The reasons for introducing the first three of these components seem to be clear. The fourth component is used to indicate whether the expert became better during the latest round.

To determine the value of a given rule/equation one can restrict the first three components to this rule/equation. So the referee computes a weighted sum for every new rule/equation he generates and delivers the best ones according to this measure.

The referee has to be fair to the expert: Experts (for example ADD-WEIGHT and GOAL-SIM) are created for totally different purposes and this has to be taken into account by the referee. This can be done by adjusting the weights to the components mentioned above.

5.3 The supervisor

The supervisor is responsible for the team meetings. He

- determines the winner for the next round
- integrates the selected results from the losers into the winner's database
- determines the new n-tuple of expert/referee pairs
- determines the time for the next team meeting.

The first task is based on the referee reports about the appropriateness of the experts in the latest round. For the integration of the results of the losers see section 3.

We give some hints to create the team for the next round in case where almost nothing is known about the problem instance. For the first rounds a standard team should be activated, including the expert ADD-WEIGHT or MAX-WEIGHT. Later on every expert should be activated periodically, he should replace the expert with the lowest rating. Additionally, if during a team meeting an expert gets a rating far below the others he should be replaced by another one. The details have to be fixed by the user.

To determine the length of a round the following rules have turned out to be useful. For the first rounds the length should be kept fixed. Next, since the database grows and henceforth it costs some time to find new useful results, the length of the rounds should grow linearly. Finally even faster growing is recommended, i.e. an exponential growth.

5.4 Implementation aspects

A crucial point with distributed systems is the need to reduce the communication overhead and the idle times of processors to a minimum. From the conceptual point of view the team work method takes this into account by limiting the communication to fixed events, the team meetings. We now discuss implementation aspects.

We have implemented the conceptual units expert, referee and supervisor as "quasi-processes" (see below). In order to minimize the transport of data on the net we in general do not send data to the quasi-processes but run the quasi-processes on that processor that has the data. So we always run an expert/referee pair on the same processor. The supervisor is active only during the team meetings. At the beginning the supervisor is run on the processor of the old winner. Here he determines the new winner for the next round. After that the supervisor is run on the processor of the new winner, here he integrates the results from the losers, determines the new team and sends the starting information to the processors of the other team members. Technically, we have implemented a single process with the three modes *expert*, *referee*, and *supervisor*. Now a quasi-process for an expert is just a process in mode *expert*. This trick allows one to realize the ideas developed above. We call this concept *floating control*.

To reduce idle times we interleave the tasks of the supervisor with the preprocessing of the team members: If an expert uses the same ordering as the new winner then he can accept the starting quadruple (R, E, g, CP) separated into these components. Otherwise he has to accept this information in the form $(\emptyset, \emptyset, g, R \cup E \cup CP)$. In any case he has to sort the CP-component according to his *choose-CP* function and that costs more time than sending data. So the supervisor first sends the CP-component of the winner without the results of the losers, he then processes the results of the losers and then sends this information to the other team members. So the time for processing the results of the old losers can be used by the new experts to preprocess their input data.

We have implemented our team work completion in C under UNIX on a cluster of SUN ELC machines. Unfortunately, up to now we have implemented the communication by message passing for a cluster of two machines only. This is the basis for the results reported in section 6. An implementation of broadcasting allowing for bigger clusters is under way.

6 Results

We will demonstrate the usefulness of the team work method on five examples from different areas of equational reasoning. Each team consists of two experts that work together. In Table 1 we compare the run time needed by the team with the sequential run time of each member of the team. The speed-up factor is the time needed by the best of the two experts divided by the time needed by the team.

The run times given in the table include the communication overhead and the idle times. So it is the time the user has to wait for the proof. For the sequential prover this is very close to the CPU-time.

example	team	1st expert	2nd expert	speed-up
Z22	5.032	16.241	39.760	3.2
div	2.813	34.698	-	12.3
luka1	15.044	95.407	40.908	2.7
luka2	13.518	23730.000	81.383	6.0
ring	307.962	-	5153.000	16.7

Table 1: run-time comparison team vs sequential experts (in seconds)

Before we comment on these results we will give brief descriptions of the examples and the teams used.

Example Z22:

Input:

$$\begin{array}{llllll} a(b(c(x))) = d(x) & b(c(d(x))) = e(x) & & & & \\ c(d(e(x))) = a(x) & d(e(a(x))) = b(x) & & & & \\ e(a(b(x))) = c(x) & a(a1(x)) = x & & & & \\ a1(a(x)) = x & b(b1(x)) = x & b1(b(x)) = x & & & \\ c(c1(x)) = x & c1(c(x)) = x & d(d1(x)) = x & & & \\ d1(d(x)) = x & e(e1(x)) = x & e1(e(x)) = x & & & \end{array}$$

Ordering: LPO with precedence $e1 > e > d1 > d > c1 > c > b1 > b > a1 > a$

Task: Complete system
Team: expert1: POLYNOM-WEIGHT
expert2: MAX-WEIGHT

The example Z22 was brought to our attention by J. A. Kalman during the CADE-10 conference. The completion of the equational system shows that the equations represent the cyclic group of order 22 (therefore the name Z22).

The system is completed by our team in two rounds. The winner of the first round is MAX-WEIGHT. POLYNOM-WEIGHT who assigns in this example to all function symbols polynomials of the form $x+c_f$ with c_f a positive number finishes the completion in the second round. The speed-up is due to the change of heuristic for choosing critical pairs because all rules selected from the results of POLYNOM-WEIGHT after the first round were already in the set of rules of MAX-WEIGHT.

Example div:

Input:

$$\begin{array}{llll} f(g(f(x))) = g(f(x)) & & c(d(b(a^3(x))) = a^3(x) & \\ h(f(g(x))) = c(e) & & a^8(x) = c(x) & \\ b(c(d(a^4(x)))) = a^2(b(c(a^2(x)))) & & b^7(x) = a(x) & \end{array}$$

Ordering: Knuth-Bendix ordering KBO with weight 1 for all symbols and precedence $h > f > g > a > b > c > d > e$

Task: Prove $c(d(b(c(c)))) = h(g^{20}(f(c)))$
Team: expert1: POLYNOM-WEIGHT
expert2: FORCED-DIV

This example shows the advantages of focusing on different parts of the set of equations. Only using the first two equations of the input $h(g^{20}(f(e))) = c(e)$ can be proved. Only using the last 4 equations $c(d(b(c(e)))) = c(e)$ can be proved. The expert FORCED-DIV can prove the right side of the goal in approx. 2 seconds and POLYNOM-WEIGHT, again only using polynomials of the form $x + c_f$ as interpretations with big c_f values for the symbols f, g and h, needs the same time to prove the left side. So, after a round of 2 seconds the expert POLYNOM-WEIGHT is the winner and gets from FORCED-DIV the rule $h(g^{20}(f(e))) \rightarrow c(e)$, which is considered very good by its referee, because it can reduce the goal. As POLYNOM-WEIGHT has already found the rule $c(d(b(c(e)))) \rightarrow c(e)$ the proof is finished.

All experts, except FORCED-DIV, generate the rule $h(g^{20}(f(e))) \rightarrow c(e)$ very late, because it is big. They concentrate mainly on the consequences of the last four input equations. Therefore they need much time until they can complete the proof (ADD-WEIGHT, MAX-WEIGHT or GOAL-SIM need the same or more time compared to POLYNOM-WEIGHT). On the other hand, FORCED-DIV concentrates on the divergence $f(g^i(f(x))) \rightarrow g^i(f(x))$ and therefore neglects the other equations. The cooperation forced by the team work method leads to an enormous speed-up by combining the strength of both experts.

Example luka1 and luka2:

Input: $C(T,x) = x \quad C(x,C(y,x)) = T \quad C(x,N(N(x))) = T$
$C(C(x,y),C(N(y),N(x))) = T \quad C(C(x,C(y,z)),C(C(x,y),C(x,z))) = T$
$C(N(N(x)),x) = T \quad C(C(x,C(y,z)),C(y,C(x,z))) = T$

Ordering: LPO with precedence $C > N > T > p > q > r$

Task: luka1: Prove $C(C(p,q),C(C(q,r),C(p,r))) = T$
luka2: Prove $C(C(N(p),p),p) = T$

Team: luka1: expert1: ADD-WEIGHT
expert2: GOAL-SIM
luka2: expert1: POLYNOM-WEIGHT
expert2: MAX-WEIGHT

The examples luka1 and luka2 are taken from [Ta56]. The input equations are an equational axiomatization for propositional calculus by Frege. Lukasiewicz gave another set of axioms of which luka1 and luka2 are the first two.

Fair sequential basic provers have problems with these examples in so far as they simply try to complete the set of input equations. The goals do not influence the computation. This is also one of the major critisims on completion based equational theorem

proving. But in our team work approach there are many concepts that force the team to concentrate on the given goal. For example, the referees take into account in their judgements reductions of the goal. Further we can include heuristics that concentrate on the goal. They are not fair, but a team strategy using them can be fair. For luka1 the winner of the first round is ADD-WEIGHT. No result of GOAL-SIM is integrated in the winning system. But in the second round GOAL-SIM completes the proof. Again, the change of the heuristic is responsible for the speed-up. GOAL-SIM is not able to generate the facts it needs for appropriate use of its heuristic. This is done by ADD-WEIGHT. For luka2 we have the same situation. POLYNOM-WEIGHT wins the first round, while MAX-WEIGHT finds no good results. But in the second round MAX-WEIGHT finishes the proof.

Example ring:

Input:

$$
\begin{array}{rcl@{\qquad}rcl}
j(0,x) & = & x & j(x,0) & = & x \\
j(x,g(x)) & = & 0 & j(j(x,y),z) & = & j(x,j(y,z)) \\
f(f(x,y),z) & = & f(x,f(y,z)) & f(x,j(y,z)) & = & j(f(x,y),f(x,z)) \\
j(g(x),x) & = & 0 & & & \\
j(x,y) & = & j(y,x) & & & \\
f(j(x,y),z) & = & j(f(x,z),f(y,z)) & & &
\end{array}
$$

Ordering: KBO with weights
$\varphi(f) = 5, \quad \varphi(j) = 4, \quad \varphi(g) = 3$
$\varphi(0) = 1, \quad \varphi(b) = 1, \quad \varphi(a) = 1$
and precedence $f > j > g > 0 > b > a$

Task: Prove $f(a,b) = f(b,a)$

Team: expert1: PREFER-RULE
expert2: ADD-WEIGHT

This example is mentioned as a challenging problem in [St84]. Reported automated proofs were obtained by using completion prover with build-in theory AC. However, our team does not use build-in theories. It needs 5 rounds to find the proof. The winner of each round is PREFER-RULE, but the proof is completed by ADD-WEIGHT. After the first round the referee of ADD-WEIGHT selects the two equations $j(x,j(y,z)) = j(y,j(z,x))$ and $j(x,j(y,z)) = j(z,j(y,x))$ that are added to the system of PREFER-RULE. Although PREFER-RULE selects no critical pairs that can not be oriented, results of other experts are considered. These equations are necessary, because they introduce the commutativity of j in the system of PREFER-RULE. (Note that although $j(x,y) = j(y,x)$ is an input equation, it will not be selected by PREFER-RULE !) The proof can only be found by ADD-WEIGHT, because the commutativity of f is needed, which will never be selected by PREFER-RULE.

So we have achieved speed-ups in very different areas – semi-thue systems (coded here by monadic function symbols), ring theory, equational propositional logic – for both

completion and proving tasks. The combination of different heuristics in a competitive but also cooperative way leads to speed-ups that are more than linear in comparison to the sequential heuristics used in the team. The example *ring* suggests that there may be other and better ways to deal with theories than using the expensive theory completion.

Note that the sequential run times are fast. Our sequential prover can compete with such systems as OTTER or REVEAL (see [Mc90] and [AA90]) on these examples. Therefore the speed-ups of the teams are not due to weakeness of our sequential prover.

7 Related work

In the literature several attempts are reported to use parallel or distributed computing to enhence the power of theorem proving. They differ in the granularity of the parallelisation and in the degree of cooperation.

Yelick and Garland [GY92] use a very fine granularity of parallelism. Their approach is based on the inference system of Bachmair, Dershowitz and Hsiang [BDH86] for the Knuth-Bendix completion procedure and the parallelisation takes place on the level of these inference rules. According to our experience this granularity is too fine. There is no aspect of cooperation and competition discussed in the paper.

On the contrary, Ertel [Er90] uses a very coarse granularity of parallelism. He uses a tableaux-based theorem prover and observes that for a fixed input problem and a fixed strategy the running time may heavily depend on the order the input data are given to the prover. So he starts parallel computations with several, randomly generated, permutations of the input data and stops as soon as one processor has found a proof. Also every decision a prover has to make is done randomly. In this approach there is no cooperation between the provers, they only compete.

Slaney and Lusk [SL90] have proposed to use parallelism to compute the clousure of a set of facts (i.e. clauses) under some inference rules. The processors share a common memory and the inferences are distributed among the processors such that each processor gets assigned facts. Then the processor generates all inferences of its facts with all other facts. The drawback with this approach is that backward subsumption (and simplification) is costly. Here only parallelism is used, there is no cooperation or competition between the processors.

The DARES system ([CMM90]) does not only distribute the process of generating new facts, but also the initial facts are distributed among the processors. Then requests have to be started to get new facts from other processors. For starting and answering such requests DARES uses heuristics and no central control is needed. In DARES the different behavior of the problem solving nodes is only achieved by different facts. No further control knowledge, like in our team work method, is used.

In a recent paper of Bonacina and Hsiang [BH92] the problem of how to guarantee fairness in distributed automatic deduction is studied. Each processor p_k has stored in its own memory at time i the set S_i^k of facts. So at time i the whole data basis consists

of $S_i^1 \cup \cdots \cup S_i^n$. Each processor works on its own data and simultaneously sends messages to draw inferences between his and foreign data. Criteria are developed that guarantee fairness (and so completeness) of the whole system. The problems here arise from the fact that the data base is distributed. In our approach, there is a common data base for all experts whenever a new round is started. This simplifies the problem to guarantee fairness. We believe that it also saves unproductive time for communication.

References

[AA90] Anantharaman, S., Andrianarivelo, N.: Heuristical critical pair criteria in automated theorem proving, *Proc. DISCO '90, LNCS 429, Springer, 1990, pp. 184 - 193.*

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

[AO83] Antoniou, G., Ohlbach, H.J.: Terminator, *Proc. 8th IJCAI, Karlsruhe, 1983.*

[BDH86] Bachmair, L., Dershowitz, N., Hsiang, J.: Orderings for equational proofs, *Proc. Symposium on Logic in Computer Science, 1986, pp. 346 - 357.*

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

[BH92] Bonacina, M.P., Hsiang, J.: On fairness in distributed automated deduction, *to be published.*

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

[De93] Denzinger, J.: TEAMWORK: A method to design distributed knowledge based equational theorem provers, (in German) *Ph.D. thesis, University of Kaiserslautern, 1993.*

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

[Er90] Ertel, W.: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems, *Internal Report TUM-19050, Technical University of Munich, 1990.*

[GY92] Garland, S.J., Yelick, K.A.: A Parallel Completion procedure for Term Rewriting Systems, *Proc. 11th CADE, 1992, pp. 109 - 123.*

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

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

[Mc90] McCune, W.W.: OTTER 2.0 Users Guide, *Technical Report ANL-90/9, Argonne National Laboratory, Argonne, 1990.*

[SL90] Slaney, J.K., Lusk, E.L.: Parallelizing the Closure Computation in Automated Deduction, *Proc. 10th CADE, LNAI 449, Springer, Kaiserslautern, 1990, pp. 28 - 39.*

[St84] Stickel, M.E.: A Case Study of Theorem Proving by the Knuth-Bendix Method: Discovering that $x^3 = x$ implies Ring Commutativity, *Proc. CADE-7, LNCS 170, Springer, 1984, pp. 248 - 258.*

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