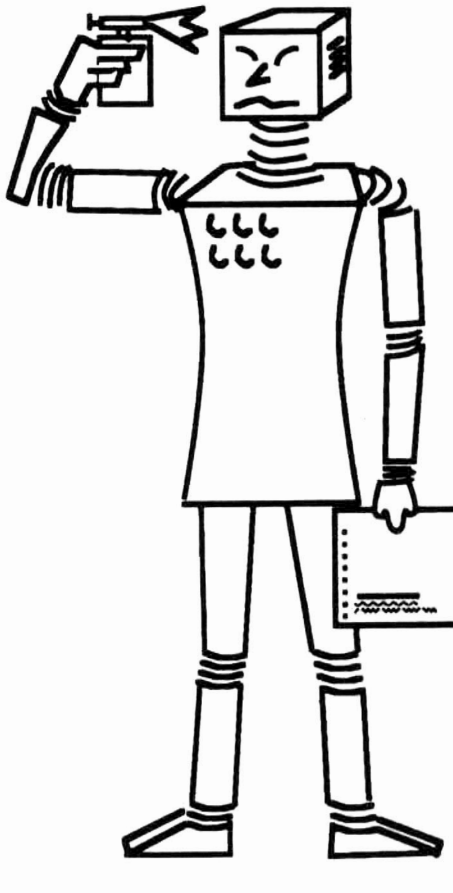
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT



Distributed knowledge-based deduction using the team work method

Jörg Denzinger

SEKI Report SR-91-12 (SFB)

Distributed knowledge-based deduction using the team work method

Jörg Denzinger
Department of Computer Science
University of Kaiserslautern
Postfach 3049
6750 Kaiserslautern
Germany
E-mail : denzinge@informatik.uni-kl.de

6. Nov. 1991

Abstract :
The team work method is a way to distribute deduction processes that are based on the generation of new facts. The key point to achieve the distribution is the use of various kinds of knowledge that is not well expressible as facts. In analogy to human project teams team work uses tactical control knowledge in form of experts, assessment knowledge in form of referees and strategic control knowledge in form of the supervisor of the team. The supervisor chooses experts that generate new facts. After a certain period of time the referees judge the generated facts and the supervisor uses the best facts to generate a better problem state for the experts.

A system using the team work method has the ability to focus on the proper problems by abstraction of results and the possibility of self-tuning of the system by reflective analysis of the computed results. The general concept of team work is instantiated to the team work completion method for equational deduction. Possible experts, referees and supervisors are discussed and their usefulness for completion is demonstrated by examples. The examples will show that super-linear speed-ups have been achieved, even for challenging problems. Implementational aspects of this approach, concerning the problem of idle times of processors, are also discussed and solutions are given.

1. **The idea** : *knowledge and distribution*

In the last years many efforts were made on the one hand side in building specialized theorem provers for special purposes and on the other hand side to investigate the usage of distributed sytems in theorem proving. Both directions should result in more efficient theorem provers. Examples for the first type of improvements are reported in [Bu85] and [KZ89] and examples for the second type can be found in [SLe90] and [SL90].

The general idea of specialized theorem provers is the availability of more knowledge in special domains of interest which leads to adjusted and more powerful methods for proving theorems in these domains. So the use of knowledge which is not expressible within the formalism of the known proving methods and the use of distributed and therefore more efficient methods offer great improvements in the area of deduction systems.

We will show in section 2 that both forms of improvements are inspired by the human way of theorem proving. And this human way shows also that the combination of both features is necessary in order to develop efficient and strong theorem provers.
The main problem in dealing with distributed systems is to find a well suited model for communication and control. In our case this model must also include the ability to use various kinds of (special) knowledge. We find such a model in the problem solving behaviour of teams in business companies, This leads to the team work method for distributed theorem proving described in section 3. In section 4 we use this method in the field of equational deduction based on completion getting a **team work completion** system.

In section 5 we give solutions to the problems which arise when our distributed system is implemented. The main point here is the question to which extend processors are idle because of the need to synchronize with other processors. We are able to show that our team work completion can overcome most of the known difficulties.

In section 6 we discuss the solutions of some examples with our system. For the well known challenge problem that a ring with $x^3 = x$ is commutative a team of just two experts could gain a speed-up factor of 11 compared to the best expert alone. Finally we sketch further improvements of our method involving the usage of results in other AI disciplines than deduction.

2. The human way of theorem proving

In contrast to other AI disciplines, like natural language understanding or intelligent tutoring systems, which are centered on understanding and simulating human methods, the most successful methods in automated deduction are normally not used by human experts in theorem proving. No mathematician would, for example, use resolution as basic inference step in his proofs. As a result of this phenomenon it was necessary to develop proof transformation techniques to get better understandable proofs out of computer generated ones ([Li90]).

Two characteristics of human proofs, namely

- the generation of a hierachy of subproblems and
- the consideration of only important steps in the proof

give us hints, why nearly all interesting proof problems can only be solved be some human beings and not by (general purpose) theorem provers.

In the following, we will give some observations about the way human experts, i.e. mathematicians, solve proof problems. Certainly this list is not complete, but characterize the use of knowledge and distribution by human beings.

1) Human experts use <u>various kinds of knowledge</u> for drawing conclusions. They do not restict to the facts that are used to formulate the problem, but they apply also some background knowledge about methods, plans, or consequences of some given facts.
2) Human experts <u>learn from old proofs and proof attempts</u>. They learn not only how to do something, but also when to do it. So beside learning new facts they also learn how to apply these facts. This results in the human ability to use analogy in proofs. The two characteristics of human proofs given above are very important to achieve this ability.
3) Human experts try to <u>break big problems into smaller</u> ones, which are hopefully easier to solve. Many special methods consist just in breaking down big problems into smaller ones and in combining the solutions of the smaller problems to get a solution for the entire problem.
4) Due to 1) and 2) human experts are able to detect whether the currently used method leads to a dead end or is not as good as expected. During a proof attempt all <u>new results are critically reviewed</u>. Note that this

ability is also for human experts not easy to achieve.

5) Different human experts have different knowledge and therefore they solve the same problem often in different ways. This means also that many proof problems can not be solved by everyone who tries it.

6) If a human expert fails in finding a proof for a (sub-) problem, because his knowledge and his methods can not help him anymore (see 4) and 5)) or if a problem is too big then he takes advise from other experts.

All these points are well known. Every mathematician is trained in using them during his education (see [Po45]).
For our goal of developing a method for knowledge based, distributed theorem proving we can formulate the following questions :

- How can different kinds of knowledge, i.e.
    - facts
    - control knowledge
    - assessment knowledge

  be represented in a system ?
- How can interaction of these kinds of knowledge can efficiently take place?
- How can distribution of the proof finding process take place, especially if it is not known a priori which subproblems have to be solved ?

The answers to the last two questions are not only important for computer systems, but also for groups of human experts who must work together to find a solution (in our case a proof) for a given problem. We have seen that human experts have the ability to work together, but how can the cooperation be enforced ?

Business companies which relay on their ability to find solutions to problems very fast and with limited resources have developed an organisational structure, the (project) team ([Ye86]), which is an answer to our questions.

A team consists of a supervisor, many experts, and some staff members, which we will later refer to as referees. The team is just put together to solve a given problem. For another problem another team would be created. The team has a limited budget, so that there can only be a limited number of team members. Normally there are more experts than places in the team, so there is a competition between the experts. Because every team member gets a reward if the problem could be solved, it is the interest of all members to cooperate.

The team works as follows : The problem is given to the supervisor who puts together a [hopefully] well suited team for this problem. Then every expert gets the problem description and tries to solve the problem alone. As stated in point 5 of our observations every expert has different methods and ideas to tackle the problem. The supervisor has to take care that the experts differ enough in their background so that not two of them use nearly the same methods.

The cooperation of the experts is enforced by the institution of team meetings. During such a meeting the results of all experts should be presented in a short form, so that these results are available for the others. Therefore not the experts give these reports, but the referees. Out of the presented results a new, more precise problem description is formed by the supervisor, so that the team can focus on the remaining problems. Note that for an expert it is not necessary to know how one of his [sub-] problem is solved, but only that this problem is solved so that he can attack other [sub-] problems. Due to the shortness of the reports only results considered by the referees and not the way they are achieved by the expert are given to the team.

The referees' task is not only to give reports about the results of experts but also to inform the supervisor about experts that are unable to contribute to the solution of the problem. Then the supervisor can exchange these experts in order to give experts with a different knowledge a chance to work on the problem. This way the limited resources are always used in a nearly optimal way.

So far, the experts are free to work on any part of the problem they think they can solve. But there are often tasks in the problem solving process which are not directly aimed at solving the problem but to detect ways the problem can definitly not be solved. Then no one other in the team should do any work on these ways. Consider for example the work of a controlling specialist in a development team who cancels ideas and solutions which are too expensive.

Other special tasks are, for example, problems for which there is only one method to solve them and the method always succeeds. In order to prevent other experts from doing this work, which would definitly result in redundancy, one special expert is selected to do the computation until the next team meeting. We will call experts doing these special tasks specialists.

The human team described above on the one hand gives the individual team members the freedom to work on problems in their individual way and on the other hand a distribution of the problem solving process is achieved. Note that the communication between the team members is regulated to have a maximum of information exchange with a minimum of time used for it. So the team is a very good model for a distributed theorem proving system.

3. The team work method for automated theorem proving

Many models for distributed problem solving, not only in AI, stem from examinations of the behaviour of groups of human beings. The general idea is always the use of processors to simulate experts ([[Sm81]]). But the tasks of these experts differ from system to system. One possibility is to have "allround" experts which are able to do every task in the system, so that the remaining problem is to get a balanced distribution of subtasks among them. A possible solution to this design problem is to use blackboard systems ([[EHLR80]]). On the other hand in some systems experts are only capable of doing very special tasks. Then the problem of "bottleneck" experts, i.e. experts whose special abilities have to be used too often, arises.

Such systems can only be used, if the subtasks to do and the subgoals to solve are known a priori or can easily be determined. But, as stated before, in theorem proving the detection of subgoals is difficult. Often a creative process is needed to find subgoals or better to find possible subgoals ([[Po45]]).

For such tasks human beings use teams. We will now simulate the behaviour of teams to develop a model for a distributed theorem proving system. Note that the implementation of the system will differ from this model in some parts in order to maintain efficiency (see section 5 for details).

The components of a system based on team work are the same as in the case of human teams : a supervisor, referees, and experts (specialists).

Experts

The task of the experts is to produce new facts and so to solve the proof problem. Because the important facts have to be shared with the other experts, all experts must use the same representation for them, for example clauses, equations, or polynomials. Although it is not necessary for the experts to use the same calculus, i.e. resolution, completion, type theory, or Gentzen calculus, we would recommend it, because then the implementation of experts is much easier.

Then the differences between the experts are established by different heuristics in choosing the next inference step. Every expert has the whole problem description, so that one expert alone works like a theorem prover controled by a heuristic. Note that the heuristic of an expert does not have to be complete, because completeness has only to be a property of the whole team. So there is a wide variety of criteria which can be used to form heuristics for experts. In section 4.2. we will present many heuristics and also some general criteria for completion a la Knuth-Bendix.

As stated in section 2, sometimes human teams use "specialists" for special purposes. In theorem proving there are such purposes, too. First, for special subproblems there can be used special solvers, eventually using different representations. An example for such a method is the simplex method ([Da63]), which can be used to find optimal solutions for systems of inequations. But the use of such special methods is very limited because the type of subproblem to solve must be known and, as stated before, in most proof problems this can not be determined easily.

Nevertheless there is a special task, which is important in nearly all theorem proving methods : The elimination of redundancies. This means to detect unnecessary inference steps before they are really performed and to detect facts which are not needed in the proof (before the proof is found).

Referees

In opposition to human teams every expert in our team theorem prover has his own referee. The tasks of our referees are the same as in human teams : First they have to judge the progress and the quality of results of an expert which results in a measure for the expert. Then they have to pick up the best results of an expert in order to transfer them to the supervisor.

So a report of a referee consists of a number [the measure of the expert] and some facts. Certainly, the main problem is how to determine this number and how to find important facts. Similar to the heuristics used by experts to determine their next inference step, heuristics using the same criteria can be built for referees. But the referees can also take into account the effects a found fact have had for an expert. This means, for example : Has the fact been used in many later inference steps, has the fact been used to remove other facts, are there any similarities of fact and a goal or has the fact only led to an immense greater search space ? In the last case the fact should certainly not be reported to the supervisor.

In order to compare experts all referees have to use the same criteria to calculate the measure of their expert. But the criteria they use to find important facts have to vary from referee to referee, because the experts can concentrate on different parts of the problem. This difference between the experts must be taken into account.

Again realisations of referees in the case of equational deduction by completion can be found in section 4.2.

Supervisor

In contrast to the supervisor of a human team, the supervisor of a system using the team work method works only during the team meetings. The supervisor receives the reports of the referees and constructs from this reports a new problem description.

Base for this description is the system of facts of the expert with the best measure. We will call this expert in the following winner. Then the facts reported by the referees of the other, loosing, experts are added to the winner's system and possible necessary operations to obtain a valid problem description are performed [In the case of completion these operations are interreduction and computing critical pairs with the new facts, see 4.2.].

Before the new problem description is passed to all experts the supervisor selects the appropriate expert/referee pairs and possibly some specialists for the next computation cycle. Again, he uses the measures of the experts to find some with bad performance and replaces them by experts from the expert database. In this database not only the expert programs are stored but they are also grouped together. So "bad" experts can be replaced by experts out of the group of experts with good performance on the current problem.

But in order to have experts with different knowledge as members of the team, all experts of the database should get periodically the chance to become a member of the team. Also in the database are informations concerning the best referees for an expert. If the supervisor selects an expert he also selects one of the possible referees for him. The supervisor controls the database and adds the measures of an expert for the current proof problem to its records.

When the supervisor has attached to each processor an expert it sends them the new problem description and gives them the time of the next team meeting. Then the new computing cycle begins.

Related work

Nearly all concepts in distributed AI (DAI) are explained by the use of analogies to human life. Very early the concept of groups of experts has been formed as foundation of DAI problem solving systems (see for example [SD81]). Later works emphasize even more on analogies to human beings ([He91]) in various situations up to modelling parts of human society, for example the scientific community ([KH81]). Also researchers in the area of social sciences have developed computer systems that simulate ways of human interaction. The TEAMWORK system ([Do85]) of Doran was designed to model the behaviour of human teams. In contrast to our approach, TEAMWORK models how human beings constitute teams in various ways. So structure and control of cooperation is not given, as in our case, but has to be established by the team members themselves.

Although there exist many approaches to DAI in general and many applications of DAI, there are only a few applications in the area of automated theorem proving. Ertel ([Er90]) examined the behaviour of systems where many identical provers work on the same problem, each prover receiving a permutation of the input facts. Compared to the use of different selection heuristics for the next inference steps (as in our team work method) these permutations are only a slight modification, but Ertel showed that the provers differed in their runtime behaviour. Ertel's approach does not contain any interaction between the provers, but emphasize, like team work, the competition aspect.

Slaney and Lusk ([SL90]) developed a general method to distribute theorem proving methods that are based on processing the closure of a set of facts under inference rules. Their main idea is to distribute the possible inference steps among the processors, which share a common memory. Each processor concentrates on the inferences of one fact with all others, and every processor uses a different fact. Although each processor makes a partial subsumption test for all facts it generates, a general subsumption test, performed by one processor for the results of all processors, is needed before new facts are added to the common memory. This approach does not take into account any further knowledge to avoid the computation of the whole closure, when only a certain goal has to be proved. Further, it is not well suited for parallelizing completion procedures, because often rules in the shared memory have to be reduced.

Finally, 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 behaviour of the problem solving nodes is only achieved by different facts. No further control knowledge, like in our team work method, is used.

4. Team work completion

In this section we instantiate the general concept of section 3 to get a distributed, knowledge-based theorem prover for equational logic. The method used is the completion method of Knuth and Bendix ([KB70]) which has proven to be useful for equational deduction. We use an extension: unfailing completion. We will discuss various experts and referees for choosing critical pairs and measuring rules and equations.

First of all, we will give a brief introduction to sequential unfailing completion.

4.1. Unfailing completion

In the following we will sketch an implementation of the inference rules for unfailing completion of [BDP89]. The components of an unfailing completion algorithm are a reduction ordering, a function to generate critical pairs and a function to compute the normalform of a term with respect to a rule system.

The reduction ordering > (see [De87] for definitions and examples of such orderings) is used to orient the equations in E into the set R of rules.

Equations which can not be ordered stay in E.

Equational consequences of E and R are generated by computing critical pairs. A critical pair of two rules or equations $l_1 = r_1$ and $l_2 = r_2$ is defined as follows : Let p be a position in l_1 which is not a variable and σ the mgu of l_1/p and l_2. If $\sigma(r_1) \not\geq \sigma(l_1)$ and $\sigma(r_2) \not\geq \sigma(l_2)$ then $\langle\sigma(r_1), \sigma(l_1[p \leftarrow r_2])\rangle$ is a critical pair between $l_1 = r_1$ and $l_2 = r_2$. Here $l_1[p \leftarrow r_2]$ denotes the term generated by replacing the term at position p in l_1 by r_2.

Rules are used to reduce (or simplify) terms in other rules or equations. Let $l \rightarrow r$ be a rule, t a term and p a position in t such that there is a substitution σ with $\sigma(l) \equiv t/p$. Then t can be replaced by $t_1 \equiv t[p \leftarrow \sigma(r)]$. We say t is reduced to t_1. This is done for all terms in all equations and rules whenever possible. If there is no rule in R which reduces t then t is called in normalform (with respect to the set of rules R, denoted by $\text{normalform}_R(t)$). If all terms in R and E are in normalform then R and E are called interreduced. For all given sets R and E interreduced sets R′ and E′ can be computed.

A completion procedure then performs the following steps with sets E and R, a reduction ordering > and a (Skolemized) goal s = t to prove :

(1) Compute all critical pairs of rules and equations in E and R. This forms the set CP.

(2) Repeat until CP = {} or $\text{normalform}(s)_R = \text{normalform}(t)_R$:

```
choose $\langle u,v \rangle \in CP$;
compare u and v with respect to > and add $u \rightarrow v$, if $u>v$ or $v \rightarrow u$, if
  $v>u$ to R or $u = v$ to E, if u and v are uncomparable;
add all critical pairs of a new rule or equation with all members of E
  and R to CP;
interreduce E and R.
```

(3) If $\text{normalform}(s)_R = \text{normalform}(t)_R$ then answer YES else NO.

A more precise procedure is later given with the procedure "expert".

The efficiency of such an algorithm is determined by the choice of the critical pairs from the set CP and the used ordering. If we look at completion runs, we detect that many of the generated critical pairs, rules, and equations are unnecessary and useless. When some very good, important rules are generated these rules, equations, and critical pairs and their consequences will be thrown away. For many examples no heuristic can generate all important rules early, but different heuristics may generate different important rules early. As we will see in the examples of section 6, the combination of heuristics, i.e. experts, by starting them in parallel and exchanging important rules can generate all important rules early. So a cooperating team of experts using these synergetic effects can be more powerful than any of the experts working alone.

There are many ways to create possibly useful experts. One expert may try to infer from the database axioms of well known structures (e.g. groups or rings) and then replace the axioms by an equivalent confluent rewrite system. But in the following we will concentrate on experts which use different heuristics for choosing critical pairs.

4.2. Distributed unfailing completion

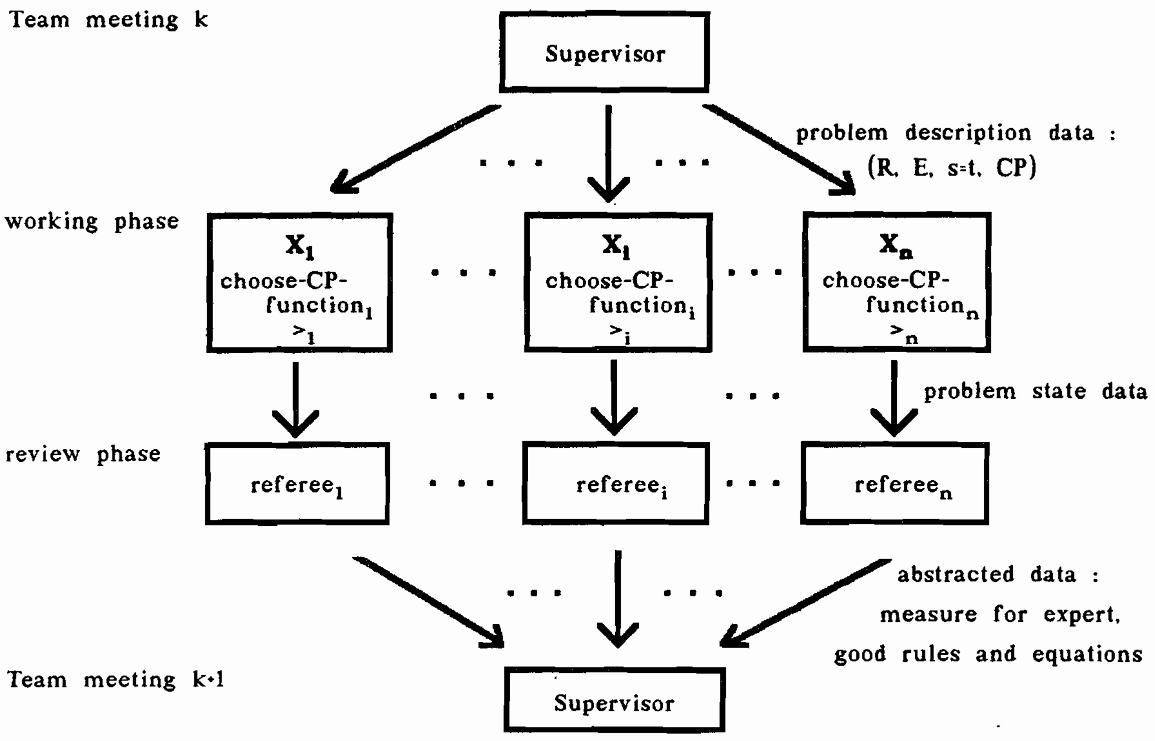
We will now instantiate all components of the team work method for the case of equational deduction using unfailing completion. Figure 1 shows the actions of this components in a cycle between two team meetings.

Experts :

Every expert executes the following procedure :

```
procedure expert :
  input : R, E, Goal $s=t$, CP, $>$, choose-CP-function
  output : YES, NO, a system (R, E, $s=t$, CP) with statistical information
  begin
    while CP $\neq$ {} do
      $(l_2,r_2)$ := choose-CP-function(CP);
      CP := CP $\setminus$ $\{(l_2,r_2)\}$;
      $l_1$ := normalform$_R(l_2)$;
      $r_1$ := normalform$_R(r_1)$;
      if $l_1 \neq r_1$ then
        if $l_1$ and $r_1$ are comparable with $>$ (let $l := \max\{l_1,r_1\}$; $r := \min\{l_1,r_1\}$)
          then R := R $\cup$ $\{l \rightarrow r\}$;
                CP := CP $\cup$ {crit. pairs between R and $l \rightarrow r$ and E and $l \rightarrow r$};
                interreduce R and E;
          else E := E $\cup$ $\{l \rightarrow r\}$;
                CP := CP $\cup$ {crit. pairs between R and $l \rightarrow r$, R and $r \rightarrow l$, E and $l \rightarrow r$ and
                              E and $r \rightarrow l$};
      if normalform$_R(s)$ $\equiv$ normalform$_R(t)$
        then answer-to-supervisor "YES";
      if interrupt-by-supervisor
        then answer-to-referee (R,E,$s=t$,CP) + statistical information;
    endwhile;
    answer-to-supervisor "NO";
  end.
```

Figure 1 : data flow in a cycle between two team meetings



The statistical information (how often a rule was successfully applied, how many rules, equations, and critical pairs are in the system, how often the goal could be reduced, etc.) is given to the referees and is used to measure the progress of the expert.

Note that, if we only have one expert in our system, then this expert performs a normal (sequential) unfailing completion with the choose-CP-function as a heuristic.

Because the experts should share their results, an expert process can not only end with the answers YES (the goal is a consequence of the initial set E) or NO, but it can also terminate when the supervisor asks for a team meeting. Of course, if one expert finds the answer YES or NO, it will interrupt all other processes because then the problem is solved or it is shown that there is no solution.

For team work completion we need many different experts. In addition to some experts that can be used for nearly every problem we must have experts with special knowledge for special problems and experts that focus on parts of the problem and phases in a proof. We have divided the experts into three groups according to the general idea used by them :

- use of syntactic arguments,

- focusing on some function symbols, thus realizing a modularization of the search space,
- focusing on special knowledge, for example aspects of the method or the goal.

For each group we will describe the currently implemented experts, but it is obvious that many more (especially in the third group) exist.

Syntactic arguments

The general idea is to weigh a critical pair by the number of symbols in its terms, a pure syntactical criterion. If we simply add the numbers of both terms and choose a critical pair with the lowest sum we get the so-called smallest-component-strategy of Huet ([Hu80]). In our system the expert using this strategy is called ADD-WEIGHT. But there are also other possible combinations of the two numbers, as for example, using the maximum of them as weight for the critical pair. We call this expert MAX-WEIGHT and, indeed, these two experts behave very differently, as documented by the examples in 6. The experts in this group only count symbols and do not give them any interpretation. So they do not use knowledge of the problem to solve. Therefore they provide a good choice for problems where no or not much (special) knowledge is available.

Focus on function symbols for modularization

Experts of this group can be used to focus on parts of the search space of a given problem. They depend on the problem and its signature. They map every symbol to a function over numbers. Then the weight of a term is the evaluation of the functions associated with the symbols in the term. If this functions are all the constant function 1, we have experts of the first group, again, without any focus.

We have implemented experts using two kinds of functions as interpretations. The first, called FWEIGHT, allows only functions f of the form $f(x_1,...,x_n) = x_1 + ... + x_n + c_f$. The other expert, called POLYNOM-WEIGHT, allows to associate polynomial functions with symbols. To achieve a focus on some symbols an expert assigns a small c_f-value or polynomial to them and large ones to all others. Since the critical pair with lowest weight is processed at any time this, indeed, realizes a focusing on the part of the search space, that is characterized by the selected symbols. The weights for FWEIGHT are easier to find than appropriate polynomials, but polynomials are better adjustable to certain parts of the problem.

An example for this kind of choose-CP-functions is reported by Stickel [[St84]], who used such a method for the completion of a ring with $x^3 = x$. In this case mathematical knowledge in ring theory determined which functional interpretation for the symbols was chosen.

Focus on special knowledge

The experts of this group focus on observations concerning the completion method, special knowledge about the given problem and its domain, or critical pairs that are similar to the goal. The focus is achieved by coding it into a weight function, as in the second group of experts. In the examples of section 6, we use three experts of this group, the FORCED-DIV expert, the PREFER-RULE expert and the GOAL-SIM expert.

The expert FORCED-DIV concentrate on a phenomenon in completion called divergence. The completion of a rule system diverges, if the final rule system is infinite. This means that the completion process will never stop. It is undecidable, wheather a given rule system will diverge or not, when completed ([He89]), but there are many approaches to avoid divergence [[He88]]. In our case, we do not want to avoid divergence, but to force it, if the goal gives hints that divergence must occur to prove it. We will discuss this expert more in section 6 in combination with example 4.

The PREFER-RULE expert also stems from knowlegde about the completion mechanism. Although we are using an unfailing completion and can therefore deal with equations, rules are preferable. So PREFER-RULE only uses equations that are explicitly given to it by the supervisor. If PREFER-RULE chooses a critical pair that is not orientable, it puts this pair back into the list of the critical pairs with a higher weight. For choosing critical pairs PREFER-RULE uses the similar criterion to ADD-WEIGHT, with the difference given above.

The expert GOAL-SIM chooses critical pairs that are "similar" to the goal. We test this similarity by using the operations unification and matching with subterms of the critical pair and subterms of the goal. We define three layers of similarity each of which contains four [sub]cases. In the following, let $s = t$ be the goal and $u = v$ a critical pair to measure.

The best critical pairs for proving a goal are certainly those which already represent a solution of the goal. This means, the goal is an instance of them, so there is a match σ with $\sigma[u] = s$ and $\sigma[v] = t$ [or vice versa]. The next best is a critical pair, such that there is a unifier μ with $\mu[u] = \mu[s]$ and $\mu[v] = \mu[t]$. If a critical pair is not similiar to both sides of the goal, then it could be similiar to one side of it, i.e. there is a match σ with $\sigma[u] = s$ or $\sigma[u] = t$ [or the same for v] or there is a unifier μ with $\mu[u] = \mu[s]$ or $\mu[u] = \mu[t]$. Note if the goal is skolemized, then the match and the unifier are the same, so that we have only two cases in this layer.

The four cases of the first layer measure only very few critical pairs. Therefore we define a second and third layer. The second layer measures such critical pairs with a part, i.e. one or two subterms, that matches or unifies with the goal. Analogously to the first layer we get the following four cases :

In the first case there is a match σ and non-variable positions p, p' with $\sigma[s] = u/p$ and $\sigma[t] = v/p'$. In the second case there is a unifier μ such that $\mu[s] = \mu[u/p]$ and $\mu[t] = \mu[v/p']$. The remaining two cases use just one side of the goal, i.e. $\sigma[s] = u/p$ or $\sigma[s] = v/p'$ or with unifiers $\mu[s] = \mu[u/p]$ or $\mu[s] = \mu[v/p']$.

Note that for Skolemized goals there are no sensible matching cases. To compare the critical pairs of one class in this layer we count all symbols in the critical pair that are not in the subterms which are instances of the goal. The bigger the subterms the greater is the similarity to the goal.

In the third layer we want to measure critical pairs that can, when ordered to rules, reduce parts of the goal. Analogously to the first two layers we get these four cases :

There is a match σ and non-variable positions p, p' [in the goal] such that $s/p = \sigma[u]$ and $t/p' = \sigma[v]$. The same with a unifier μ : $\mu[s/p] = \mu[u]$ and $\mu[t/p'] = \mu[v]$. In these two cases a reduction with the ordered critical pair leads to a new goal with sides that are not so different as before, because both sides of the critical pair are similar with the goal. Finally we have the two one-sided cases : $s/p = \sigma[u]$ or $t/p' = \sigma[u]$ and $\mu[s/p] = \mu[u]$ or $\mu[t/p'] = \mu[u]$.

To compare critical pairs in on of the classes of this layer we now count the number of symbols in the goal except the number of symbols of the subterms that are used to establish the similarity. Critical pairs that are in no layer will get such a weight that they are only chosen when there is no critical pair that is in one of the layers.

The expert GOAL-SIM uses knowledge that is very important towards the end of a proof. If we look at human proofs for equalities we can normally detect that the size of the derived terms increases at first and then has to be reduced back (after some changes). Our expert is very good in reducing back, but the big terms (critical pairs) have to be generated by an other expert of the team. GOAL-SIM can play a similar role for equational deduction by completion as the terminator for connection graphs does for resolution (see [Ko75] for a description of the connection graph and [AO83] for the description of the terminator).

There is a wide variety for other pure knowledge-based experts. Especially the works of Suttner ([Su89]) and a group of students of Ultsch ([Ul90]) give hints how successful proofs can be used to learn experts (i.e. heuristics) for other analogous problems. Like the expert GOAL-SIM such experts need the assistance of other experts to solve (sub)problems that are not similiar to the learned ones. Team work provides an ideal base for the use of such experts.

We also mentioned a special kind of experts, called specialists, which perform specialized tasks for the problem solving process. The main difference between experts and specialists is that specialists can not solve the whole problem alone. There is a task in unfailing completion, which is perfectly suited for such specialists. This is the detection of "uncritical" critical pairs. Before a critical pair is transformed into a rule or equation, it is checked if the normalforms, with respect to the current system, of both terms of the critical pair are identical. If this is the case, the critical pair is of no use and deleted.

If we look at the algorithm "expert", then we see that already the experts detect "uncritical" critical pairs by generation of normalforms. In addition, the current rule and equation systems of the experts change. However, if we consider challenging problems with huge numbers of critical pairs then all experts only examine a small part of the set of critical pairs between two team meetings. A specialist searching for unnecessary critical pairs can examine much more critical pairs than an expert whose main task is to generate new knowledge, i.e. new rules and equations.

In the presence of equations, especially the commutativity f(u,v) = f(v,u), very often a critical pair is generated several times. So a check for identical critical pairs can reduce the use of the expensive (in terms of time) normalform generation.

These two tasks, i.e. testing critical pairs for identity and equal normalforms, are performed by our REDUCE—CP specialist, which reports the unnecessary critical pairs directly to the supervisor. Again, the usefulness of such a specialist is reported in section 6.

There are some criteria, that define another kind of unuseful critical pairs, for example the criterion in [KMN88]. These criteria can also be used by specialists.

Referees :

As stated before, referees have to fullfil two tasks. First they have to judge the behaviour of their experts on the current problem since the last team meeting. Second they have to extract "good" results, i.e. rules and equations, from the loosing experts for the next computation round.

For the first task all referees use the same method. They compute a weighted sum of the following components :

- the number of rules,
- the number of equations,
- the number of critical pairs,
- the number of reductions of the goal(s),
- the number of reductions made in the last computing period,
- the average weight of all chosen critical pairs in relation to the average weight of the last critical pairs chosen by the expert.

The higher the last result the better is the expert performing at the measuring time (Remember that the experts always choose a critical pair with minimal weight !). Additionally, a high reduction rate indicates a good performance of an expert. In opposition to this the more rules, equations, and especially critical pairs are in the current system of an expert the worse should this expert be rated. But there are also examples, where it is necessary to get many new equations and rules, so that the last statement is relative. For such examples the last result alone (relation latest critical pairs to all chosen critical pairs) is well suited for judgements of the experts.

For the second task we have developed various methods to be used by our referees. The first, simply called LAST, chooses the last n rules and equations generated by an expert. If the performance of this expert is not bad, and it can thus be expected that the expert will be in the team for further computation periods, these last rules and equations enable the expert to continue his work in the next period. This is very important for the expert FORCED-DIV.

Referees can also use the same methods for choosing rules and equations as those that experts use to choose critical pairs. But they can also take the consequences of the rule or equation for the whole system into account. Therefore we have referees GOAL-TEST and STATISTIC. GOAL-TEST uses the same method as the expert GOAL-SIM. This referee should certainly not be used for the expert GOAL-SIM, but for other experts it is well suited.

The referee STATISTIC judges rules and equations by their effects on the other rules and equations. These effects of a rule or equation are :

- the total number of reductions made with it,
- the number of reductions of left hand sides of rules made with it,
- the number of reductions of equations made with it,
- the number of generated critical pairs with it.

For some examples, when many equations are generated, rules that reduce equations are more important than rules that can only be used to throw away critical pairs by reduction. Note that equations generate more critical pairs than a rule, because both sides of it are used. Also, if one is not so interested in theorem proving, but in pure completion, rules that reduce other rules (i.e. left hand sides) are important.

In the examples of section 6 we only used some of these referees. In addition, there are many more methods referees can use to judge rules and equations. They can combine some of the previously mentioned methods, for example. If subgoals that are needed to prove a goal are known a priori, the GOAL-TEST referee can also measure the similarity to these subgoals. The same ideas to learn expert heuristics can be used to learn referee heuristics.

Note that different referees should judge the overall behaviour of an expert the same thus giving an objective judgement, but that they will not choose the same rules and equations, because this is done in a very subjective way. So choosing the right referees is important for the performance of the whole team.

Supervisor :

The supervisor is responsible for the team meetings. Using the reports of the referees he

- determines the best expert,
- collects the rules and equations from all other experts,
- forms the new problem to prove,
- chooses experts and referees for the next computation period and
- determines, when the next team meeting will take place.

The first two tasks are easily performed, because the referees give the supervisor all the necessary information. In order to form the new problem to be proved the supervisor takes the current system of the best expert (i.e. the set of rules, the set of equations and the set of critical pairs) and adds the rules and equations chosen by the referees of the other experts. When adding these new rules and equations, ordered by the reduction ordering used by the best expert, it must interreduce the rule system and the equations. Then it has to add all critical pairs of the new rules and equations with the old ones and with themselves. In addition, the results of the specialists have to be used to remove the unnecessary critical pairs. The resulting system must be transfered to all experts. In section 5 we demonstrate, how these operations can be done efficiently by an interleaving technique.

There are three possible criteria for the supervisor, when he wants to replace experts. First, at least every n_1 cycles, there should be a specialist in the team. The supervisor exchanges the expert with the worst current result. with this specialist. Second, every n_2 cycles, a new expert should get the chance to participate in the team. Again, for the new expert, the expert with the worst current result will be fired. Finally, the supervisor must have the ability for fast reaction. Therefore experts, who have had the n_3 last cycles a result, which is less then k percent of the result of the winning expert, have to be replaced.

The parameters n_1, n_2, n_3, and k are given to the supervisor by the user. At the moment, in our implementation the next chosen expert or specialist is the next in the expert database. If the supervisor is at the end of the database he continues at the beginning, again. This way all experts have a chance to be chosen. The more knowledge about the experts is accessible to the supervisor the better heuristics for choosing an appropriate expert for the current problem can be used.

Finally, the supervisor determines the length of the periods between team meetings. Again, there are three possibilities in our team work completion system. First, the supervisor always uses periods of the same length. As the time needed to generate a new rule or equation grows with growing number of rules and equations (remember that all critical pairs with this new rule have to be generated), this first method is only well suited for small examples. A second possibility are linearly growing periods. Then the length of the n-th period is n times the length of the first period. The last method, which should be used for large examples, is to choose exponentially growing time periods between the team meetings.

For future versions of our system we plan to use variable time periods according to the reports of the referees. If the supervisor thinks he has a good team then he should give them more time for computing new facts than for a bad or unexperienced team. But the time period must be long enough for every expert to get some results. Otherwise a judgement of the experts is impossible.

5. The implementation : *Comments on communication and control*

The main problem arising from distributed systems is the effort to install a control in order to solve the given problem by interaction between all processors. Centralized as well as distributed control approaches need a certain amount of communication between the different processes. While a process, that is in our approach an expert, referee or the supervisor, sends information to or waits for information from other processes, it can not contribute to the solution of the problem. So one goal in developing distributed systems is to reduce the amount of communication and hence the idle times of the processors.

The structure of our team work completion approach supports this goal, because communication between processes is restricted to certain periods. During its work an expert process needs no communication. After an iteration of the while-loop (in procedure "expert") the expert checks, whether a team meeting will happen soon. If not, it immediately starts another iteration.

So the important point is the communication (1) between supervisor and experts while sending the new problem description after a team meeting,

[2] between an expert and its referee and, finally, [3] between the referees and the supervisor. Also when the supervisor is preparing a new computation cycle all the other processors are idle.

We will now discuss implementational and structural issues which reduce the loss of efficiency by processor idle times.

First of all, the idea to place referees between the experts and the supervisor in order to achieve abstracted reports of the results of the experts drastically reduces the one-to-one communication between processes, if we let experts and their referees be realized on the same processor. This means that on a structural level we distinguish between an expert and a referee, but in the implementation there is one process with two functions as arguments. The choose-CP-function realizes the expert's behaviour and a referee-function, which works on the same memory as the expert, realizes the referee. By this way we achieve an overlap of these logical processes without any communication between processors.

In order to send the system of the winning expert of the last period to all experts the supervisor process has to get this system from this best expert. As we want to solve big problems the transfer of this data would take a long time which results in idle times of all other processors. So we decided to install a floating control and therefore to let the winning expert process become the new supervisor.

This means we only have one type of process working in three modes : expert, referee, and supervisor. If the old supervisor process decides which expert will be the new supervisor, i.e. the winner, it passes the best rules and equations of all loosers to the new supervisor. Also the statistic information about the experts will be sent to the new supervisor. Note that for big problems the size of all this data is much smaller than the size of a whole system consisting of many rules, equations, and critical pairs.

There remains the problem to transmit the new system to all experts. The trick to solve this problem is inspired by experiences with the implementation of sequential completion procedures. If someone always has to take the smallest element out of a set, he will implement the set as an ordered list in which the first element is the smallest. So every expert realizes the set CP as an ordered list where the order is determined by the choose-CP-function. When the supervisor transmits the new system of rules, equations, and critical pairs via broadcast [i.e. to all processors in parallel], the experts can insert every transmitted critical pair in their CP-list according to the measures with respect to their own choose-CP-functions. That clearly results in more efficiency. Because the supervisor has to send the critical pairs one by one, it can use the information generated by the specialists to cancel unnecessary critical pairs. Every critical pair that is considered unnecessary will simply not be sent by the supervisor.

If we use whole computers instead of processors as nodes in our system, then, with the use of large buffers, the supervisor can transmit the critical pairs faster than the other nodes can receive and insert them. This time can be used by the computer representing the supervisor to interreduce rules and equations of its own system with the result rules and equations reported by the referees. If the data is transmitted in the following order :

critical pairs of the supervisor,

interreduced set of rules and equations,

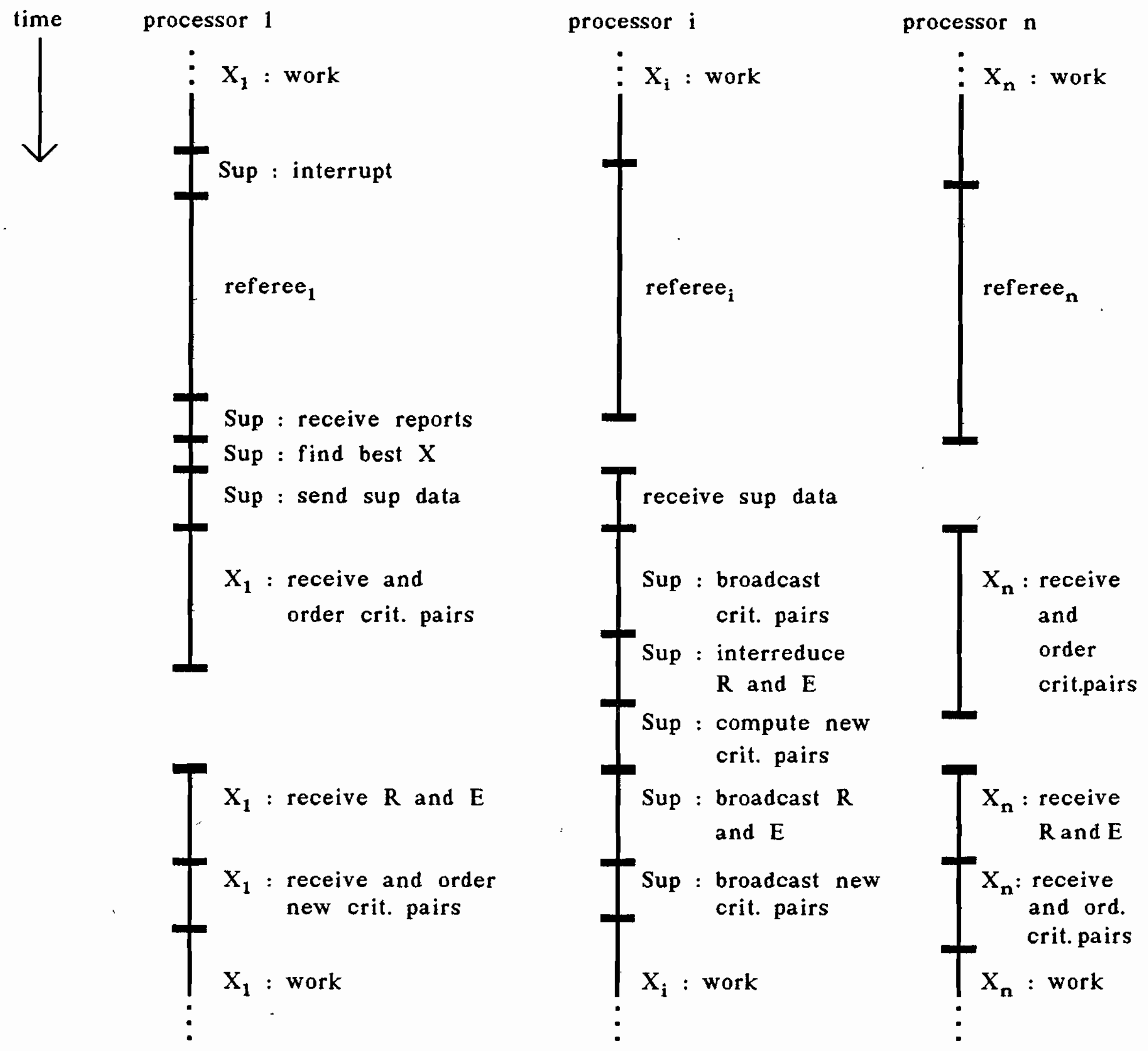
new critical pairs generated with added rules and equations,

then we nearly have no idle times of any processor during the communication time.

In addition, since the main part of the supervisor's work is exactly to compute the new system and to send this system to the experts, the idle time of the nodes, on which the supervisor process is not running, is reduced to the time the supervisor needs to choose its successor and to send the necessary informations to it.

Considering these issues our implementation should show a behaviour of the nodes of our team work completion system in a cycle between two team meetings as outlined in figure 2. Unfortunately, in our current implementation the operating system does not allow to use broadcasting for sending the new problem system to all processors. Therefore the data has to be sent to all processors one by one. This has to be taken into account when judging the computation times for the examples presented in section 6. Using broadcasting we hope to get better times.

Figure 2 : processor engagement before, during and after a team meeting



6. Experiments

As stated in chapter 1, a main research goal in automated deduction is to improve the efficiency of automated theorem provers. But what is the efficiency of a prover? As we want to prove more and harder problems, one aspect of efficiency is the time a prover needs to find a solution. If the prover needs plenty of time to solve small examples we can not expect it to solve the challenging problems. Because human beings use the prover, not the processor time used to find the solution, but the time he waits for the solution, i.e. real time, has to be clocked. Therefore table 1 compares the real time results of each expert solving our examples alone (or not) with the results of a user chosen good team. When we describe the examples we will also describe the chosen team and its behaviour while solving the example.

Example	team	ADD-WEIGHT	MAX-WEIGHT	FWEIGHT	GOAL-SIM	FORCED-DIV
1	9.5	29.93	40.18	17.14	-	-
2	10.6	115.08	58.41	89.57	-	-
3	15.84	-	167.52	-	63.89	-
4	9.0	94.75	29.0	19.16	-	-

Table 1 : Comparison of a selected team with single experts, runtime in sec

But there is another aspect of efficiency. Todays automated theorem provers offer the user many ways to influence their problem solving behaviour (see for example the Otter system [Mc88]). Many examples only can be solved, when special heuristics and parameter adjustments are provided by the user. An experienced user is able to find the right heuristics and adjustments with just a few tries. We made such an adjustment by starting the team work completion with given experts to get the times of table 1.

Unexperienced users or users with no background in the method used by the prover have nearly no chance to use these provers for their problems. Its potential efficiency in terms of runtime is of no use. In our team work completion system the supervisor enables such users to solve their problems. Starting each example with a fixed team the supervisor adjusts the team with every team meeting more and more to the given problem. In table 2 we see that this kind of self-tuning sometimes results in needing more time to find a solution even compared to the best expert. But the important point here is that the system automatically finds for each example a solution. We remark that no expert alone is able to solve all our examples in an appropriate time, say within 10 min for examples 1-4, and within 5 h for example 5, but both teams have solved them within this time.

Example	best team	standard team	best expert
1	9.5	9.5	17.14
2	10.6	15.02	58.41
3	15.84	29.52	63.89
4	9.0	25.04	19.16

Table 2 : Comparison best team / standard team / best expert, times in sec

We implemented team work completion in C under Unix. Our results were achieved on a cluster of Sun 4 computers.

We now comment the examples in detail : The first four of our examples origin from the domain of monadic function symbols. This means, we only have function symbols with arity 1 and (in goals) Skolem constants. For this domain of interest we were able to include special knowledge in some of our experts, namely in the expert FWEIGHT. We used a cluster of 2 processors. In the following we use $a^n(x)$ as abbreviation for $a(a(....(x)...)$, i.e. n times the function a is applied.
Note that the expert PREFER-RULE is identical to ADD-WEIGHT for examples 1-4, because during the completion of these examples all critical pairs are orientable.

Example 1 : The group Z22, i.e. the cyclic group with 22 elements

Completion of

$$\begin{array}{lll} a(b(c(x))) = d(x) & b(c(d(x))) = e(x) & c(d(e(x))) = a(x) \\ d(e(a(x))) = b(x) & e(a(b(x))) = c(x) & a(A(x)) = x \\ A(a(x)) = x & b(B(x)) = x & B(b(x)) = x \\ c(C(x)) = x & C(c(x)) = x & d(D(x)) = x \\ D(d(x)) = x & e(E(x)) = x & E(e(x)) = x \end{array}$$

using an LPO ([KL80]) with precedence

$E > e > D > d > C > c > B > b > A > a.$

The completed rule system is

$$\begin{array}{lll} E(x) \to a^{17}(x) & e(x) \to a^{5}(x) & D(x) \to a^{19}(x) \\ d(x) \to a^{3}(x) & C(x) \to a^{7}(x) & c(x) \to a^{15}(x) \\ B(x) \to a^{13}(x) & b(x) \to a^{9}(x) & A(x) \to a^{21}(x) \\ & a^{22}(x) \to x. & \end{array}$$

Our team of table 1 consists of the ADD-WEIGHT and the MAX-WEIGHT expert. The referees of both are STATISTIC ones. This is also the starting team for table 2, which was able to solve all examples.

For example 1 the team needed 3 cycles. The best expert of the first cycle was MAX-WEIGHT, of the second ADD-WEIGHT and in the third the solution was found by ADD-WEIGHT. After the first cycle the referees of both experts have chosen a rule that eleminates the function c, but on different ways. Starting with the system of MAX-WEIGHT after the first team meeting

ADD-WEIGHT needed only 153 rules to complete the system. Without this help by MAX-WEIGHT it needed 187 rules. The reason for the speed-up lies in the change of the heuristic for choosing critical pairs after a certain time period, which results in fewer steps.

Note that the good result of expert FWEIGHT is based on the knowledge, that all functions can be expressed in terms of the function a. Therefore FWEIGHT used a c_a-value of 1 and a c_f-value of 4 for all other function symbols. The use of FWEIGHT in the team leads to nearly the same time as the added time of the processors of the team reported above.

Example *2* :

Completion of

$$\begin{aligned}
a^{10}[x] &= b[a[b[a[b[x]]]]] & a^2[b[a[b^3[x]]]] &= b[a[b^2[x]]] \\
b[a^9[b[x]]] &= b[a[b[a[b[a[b^2[x]]]]]]] & b[a^8[b[x]]] &= b[a[b[a[b^3[x]]]]] \\
b^6[x] &= b[a[b[x]]] & b^4[a^2[b[a[b[x]]]]] &= b[a[b^4[x]]] \\
b^5[a[b[x]]] &= b[a[b^5[x]]] & b^3[a[b[x]]] &= b[a[b^3[x]]] \\
b^5[x] &= a^2[b[a[b[x]]]] & b^4[a[b[x]]] &= b[a[b^4[x]]] \\
a^2[b^2[x]] &= b[x] & a^2[b[a[b[a^2[b[a[b[x]]]]]]]] &= b[a^3[b[a[b[x]]]]] \\
b[a[b[x]]] &= b[a^2[b[a[b[x]]]]] & b^3[a^2[b[a[b[x]]]]] &= b[a[b^3[x]]] \\
b^2[a[b[x]]] &= b[a[b^2[x]]] & b^2[a^2[b[a[b[x]]]]] &= b[a[b^2[x]]] \\
b^4[x] &= a^4[b[a[b[x]]]] & a^2[b[a^3[b[a[b[x]]]]]] &= b[a[b^4[x]]] \\
a^2[b[a[b^4[x]]]] &= b[a[b^3[x]]] & b[a[b[a[b^4[x]]]]] &= b[a^6[b[x]]] \\
b[a[b[x]]] &= a^2[b[a[b^2[x]]]] & b[a^3[b[a[b[x]]]]] &= a^2[b[a[b[a[b[x]]]]]] \\
b^2[a^3[b[a[b[x]]]]] &= b[a[b[a[b[x]]]]] & b[a[b[a[b[a[b^3[x]]]]]]] &= b[a^7[b[x]]] \\
b^3[a^2[b[a[b[x]]]]] &= a^2[b[a[b^4[x]]]] & a^2[b[a^2[b[a[b[x]]]]]] &= a^2[b[a[b[x]]]] \\
b^4[a^3[b[a[b[x]]]]] &= b[a[b[a[b^3[x]]]]] & b^3[a^3[b[a[b[x]]]]] &= b[a[b[a[b^2[x]]]]] \\
b^4[a^2[b[a[b[x]]]]] &= a^2[b[a^3[b[a[b[x]]]]]] & a^2[b[a[b[a[b^2[x]]]]]] &= b[a[b[a[b[x]]]]] \\
b^2[a^2[b[a[b[x]]]]] &= a^2[b[a[b^3[x]]]] & a^2[b[a[b[a[b^4[x]]]]]] &= b[a[b[a[b^3[x]]]]] \\
b[a^2[b[a[b[x]]]]] &= a^2[b[a[b^2[x]]]] & a^2[b[a[b[a[b^3[x]]]]]] &= b[a[b[a[b^2[x]]]]]
\end{aligned}$$

$$b[a[b[a\ [b[a[b[x]]]]]]] = a\ [b[a[b[a[b[a[b[x]]]]]]]]$$

$$a^2[b[a[b[a[b[a[b[a[b^2[x]]]]]]]]]] = b[a[b[a[b[a[b[a[b[x]]]]]]]]]$$

$$a^2[b[a[b[a[b[a[b^3[x]]]]]]]] = b[a[b[a[b[a[b^2[x]]]]]]]$$

$$a^2[b[a[b[a^3[b[a[b[x]]]]]]]] = b[a[b[a[b^4[x]]]]]$$

using an LPO with precedence

$b > a$.

The completed rule system consists of the following rules :

$$\begin{array}{lll}
b^2[x] = a^{28}[b[x]] & b[a[b[x]]] = a^{20}[b[x]] & b[a^2[b[x]]] = b[x] \\
b[a^3[b[x]]] = a^{22}[b[x]] & b[a^4[b[x]]] = a^2[b[x]] & b[a^5[b[x]]] = a^{24}[b[x]] \\
b[a^6[b[x]]] = a^4[b[x]] & b[a^7[b[x]]] = a^{26}[b[x]] & b[a^8[b[x]]] = a^6[b[x]] \\
b[a^9[b[x]]] = a^{28}[b[x]] & b[a^{10}[x]] = a^{10}[b[x]] & a^{30}[b[x]] = b[x]
\end{array}$$

Our team consists of the experts ADD-WEIGHT and FWEIGHT. FWEIGHT centers on the symbol a, i.e. $c_a = 1$, $c_b = 4$. Again both referees are STATISTIC ones.

This team solved the example in 2 cycles. The winning expert of the first cycle was ADD-WEIGHT. As in example 1 the loosing expert, i.e. FWEIGHT, was then able to finish the completion.

The bad results of the single experts are due to the generation of rules with big left hand sides, which can only be reduced lately. So all their critical pairs have to be generated and later thrown away.

Example 3 :

Proof of the goal

$$E[i] = i$$

with the equations

$$\begin{array}{lll}
b[a[x]] = a[b[c[x]]] & c[a[x]] = a[c[d[x]]] & c[b[x] = b[c[e[x]]] \\
d[a[x]] = a[d[x]] & d[b[x]] = b[d[x]] & d[c[x]] = c[d[x]] \\
e[a[x]] = a[e[x]] & e[b[x]] = b[e[x]] & e[c[x]] = c[e[x]] \\
e[d[x]] = d[e[x]] & b^5[x] = c[x] & c^5[x] = d[x] \\
d^5[x] = e[x] & A[a[x]] = x & a[A[x]] = x \\
B[b[x]] = x & b[B[x]] = x & C[c[x]] = x \\
c[C[x]] = x & D[d[x]] = x & d[D[x]] = x \\
E[e[x]] = x & & e[E[x]] = x
\end{array}$$

using an LPO with precedence

$$E > D > C > B > A > e > d > c > b > a.$$

The team of table 1 consists of the experts GOAL-SIM and MAX-WEIGHT with STATISTIC referees. It solved the problem in 2 cycles. The best expert of the first cycle was MAX-WEIGHT and it received 3 rules from GOAL-SIM. One of these rules was $a[b^6[x]] \rightarrow b[a[x]]$, which it used to generate the necessary rule $E[x] \rightarrow x$ at once. In contrast to example 1, where the change of the heuristic is responsible for the speed-up, here the results of the loosing expert improve the system.

Example 4 :

Proof of the goal

$a^{30}(b(e)) = h(G^5(f(e)))$

with the equations

$$G(x) = g^5(x) \qquad f(g(f(x))) = g(f(x)) \qquad h(f(g(x))) = b(e)$$
$$b(a^{10}(x)) = b(a(b(a(b(x))))) \qquad b^6(x) = b(a(b(x))) \qquad a^2(b^2(x)) = b(x)$$

using a LPO with precedence

$G > h > f > g > b > a > e.$

Our team consists of the experts FORCED-DIV and FWEIGHT with referees LAST and STATISTIC. In fact, this example combines two subproblems. The first, represented by the first three equations is to show that $h(G^5(f(e)))$ is equal to $b(e)$. To prove this, expert FORCED-DIV was appropriate. The second subproblem is to prove $a^{30}(b(x))$ is equal to $b(x)$. To quickly solve this problem, we have to fade out all critical pairs with G, h, f, or g in it. Therefore we used expert FWEIGHT with $c_G = c_h = c_f = c_g = 100$ and $c_b = c_a = c_e = 2$. After the first computation period the referees of both experts reported the solution of the subproblems to the supervisor. It found the solution of the whole problem while generating the new problem description.

Note that the good result of expert FWEIGHT alone is due to the fact that with the solution of the second subproblem the system resulting out of equations 4 to 6 is completed and therefore there are no critical pairs without G, h, f or g. So FWEIGHT solves subproblem for subproblem without any interference between them. Our standard team has to deal with such interferences and therefore performs so bad.

Finally, we have tested team work completion on a challenge problem, the proof of the commutativity of a ring with added axiom $x^3 = x$ ([St84], [LO85]).

Example 5 :

Proof of the goal

$f(a,b) = f(b,a)$

with the equations

$j[0,x] = x$ $j[x,0] = x$ $j[g[x],x] = 0$

$j[x,g[x]] = 0$ $j[j[x,y],z] = j[x,j[y,z]]$ $j[x,y] = j[y,x]$

$f[f[x,y],z] = f[x,f[y,z]]$ $f[x,j[y,z]] = j[f[x,y],f[x,z]]$ $f[j[x,y],z] = j[f[x,z],f[y,z]]$

$f[x,f[x,x]] = x$

using a KBO ([KB70]) with weights

f : 5, j : 4, g : 3, 0 : 1, b : 1, a : 1

and a precedence

$f > j > g > 0 > b > a$.

Our team consists of the experts ADD-WEIGHT and PREFER-RULE and the specialist REDUCE-CP which was member in the team every forth cycle. The results of this team, the standard team and the experts are collected in table 3. The proof was found in the sixth cycle by expert ADD-WEIGHT. But the winning expert was always PREFER-RULE. REDUCE-CP was able to detect 142 "uncritical" critical pairs that were, at this time, not detected by the experts. This example also shows that it is not necessary to use the "expensive" theory completion method, as Stickel, to find a proof.

Example	best team	standard team	ADD-WEIGHT
5	702.12	11212.38	8310.11

Table 3 : Comparison best team / standard team / best expert, times in sec for the ring example

These experiments show the usefulness of the team work method. For all examples a linear speed-up has been achieved, at least. The method is well suited for both completion (examples 1-2) and equational theorem proving (examples 3-5). Although the time used by the expert ADD-WEIGHT to prove example 5 is not acceptable, the time used by our team without the usage of build-in theory AC or other improvements (see [KZ89]) is acceptable for a general purpose prover.

Due to their strategy for choosing critical pairs the experts POLYNOM-WEIGHT, GOAL-SIM, FORCED-DIV, and some versions of FWEIGHT can only be successfully used in a team. The referee STATISTIC was able to solve the difficult problem of choosing good results of the experts. Especially in example 5 the concept of regularly judging generated facts by a referee, has cut down the number of unnecessary commutative versions of rules and equations.

The idea of some experts generating a wide variety of new facts and other experts only concentrating on the consequences of a few facts, shows its usefulness n all examples. The best experts during the first meetings did not solve the problems, but the loosing ones, using the system of the winning expert.

7. The conclusion : *A beginning and the future*

We have presented a distributed approach to equational reasoning by team work completion. Based on the problem solving behaviour of project teams we have developed the team work method as a system with three classes of components : experts, referees and a supervisor. Every component in the system realizes a different view on the problem to solve or on a subproblem. The cooperation of all components is realized by the concept of team meetings.

Each class of components uses knowledge to fulfill its tasks. Experts use knowledge to generate new facts and to focus on parts of all the facts. Referees use knowledge to judge the work of experts and therefore provide an abstracted view on their work. Due to this abstraction the whole system has the ability to forget useless and unnecessary results and facts. The supervisor leads the search of the experts into interesting directions according to the results of the referees. Further, it has the ability to re-configurate the system in a reflective way, so that parts of the accumulated knowledge of the system can temporaly be faded out.

Our system includes aspects of the areas learning, reflective systems, planing, knowledge representation, and deduction in a natural way. We feel the need of combining aspects of all these areas, because we believe that the human ability to solve very different problems in all parts of human interest is based on an interaction of all these methods (and perhaps more than these).

As an instantiation of this general concept we mainly focus on equational reasoning. There the influence of the mentioned areas is not very strong up to now. That means we only use as many of the results in these fields as can easily be added or follow from our concept. So there is a wide variety of improvements of our systems according to results of these areas, such as

- learning expert heuristics from successful proofs
- learning referee heuristics for special domains of interest
- representing knowledge about experts/referees
- choosing appropriate experts or expert sequences by planning

etc.

Further, the team work completion approach allows, as the team work method in general, the use of knowledge or better enforces it. As there are many domains of interest with different knowledge where equational deduction is needed there has to be a huge number of new experts dealing with this knowledge. Note that our system provides a way to deal with huge numbers of experts of general interest and for special domains. So we do not have to generate specialized versions of it or specialized control structures for different domains.

For the area of equational deduction by completion team work overcomes the often heard opinion, that a forward chaining process like completion can in no way be goal oriented. The expert GOAL-SIM enables our teams to work goal directed and contributes to faster proofs (see example 3).

Our experiments have shown the usefulness of team work in an impressive manner. In our examples, our teams have always been faster than the best expert working alone. A team using two processors has needed only 54% downto 8% of the time of the best expert.

Nevertheless the system presented here is only the starting point for many more experiments and improvements in distributed completion. However, other deduction mechanisms can be distributed in the presented manner as well, if they can be placed in the team work szenario.

Acknowledgement:

I would like to thank Prof. Dr. J. Avenhaus and my collegues B. Gramlich, J. Kammer, A. Präcklein, B. Reinert, and I. Sonntag for their comments on earlier drafts of this paper and W. Pitz for his excellent implementation of team work completion.

References:

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

[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.

[Bu85] Bundy, A. :
Discovery and Reasoning in Mathematics,
DAI Research paper No. 266, University of Edinburgh, 1985.

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

[Da63] Dantzig, G.B. :
Linear Programming and Extensions,
Princeton University Press, Princeton, 1963.

[De87] Dershowitz, N. :
Termination of Rewriting,
J. Symb. Computation 3, Academic Press, 1987, pp. 69-116.

[Do85] Doran, J. :
The Computational Approach to Knowledge, Communication and Structure in Multi-Actor Systems,
Survey Conference on Sociological Theorey and Method 3, Aldershot, 1985.

[EHLR80] Erman, L.D. ; Hayes-Roth, F. ; Lesser, V.R. ; Reddy, D.R. :
The Hearsay-II Speech-Understanding System : Integrating Knowledge to Resolve Uncertainty,
Computing Surveys, vol. 12, no. 2, 1980.

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

[He88] Hermann, M. :
Vademecum of Divergent Term Rewriting Systems,
CRIN 88-R-082, Vandoeuvre, 1988.

[He89] Hermann, M. :
Crossed Term Rewriting Systems,
CRIN 89-R-003, Vandoeuvre, 1989.

[He91] Hewitt, C. :
Open Information Systems Semantics for Distributed Artificial Intelligence,
Artificial Intelligence 47, 1991, pp. 79-106.

[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. Leach, Pergamon Press, 1970, pp. 263-297.

[KH81] Kornfeld, W.A. ; Hewitt, C. :
The Scientific Community Metaphor,
IEEE Transactions on Systems, Man, and Cybernetics 11, 1981, pp. 24-33.

[KL80] Kamin, S. ; Levy, J.-J. :
Attempts for generalizing the recursive path orderings,
report, Department of Computer Science, University of Illinois, 1980.

[KMN88] Kapur, D. ; Musser, D.R. ; Narendran, P. :
Only Prime Superpositions Need be Considered in the Knuth-Bendix Completion Procedure,
Journal of Symbolic Computation 6, 1988, pp. 19-36.

[Ko75] Kowalski, R. :
A Proof Procedure Using Connection Graphs,
Journal of ACM, vol. 22, no. 4, 1975.

[KZ89] Kapur, D. ; Zhang, H. :
A Case Study of the Completion Procedure : Proving Ring Commutativity Problems,
Tech. Rep. 89-25, University at Albany, 1989.

[Li90] Lingenfelder, C. :
Structuring Computer Generated Proofs,
Ph. D. Thesis, University of Kaiserslautern, 1990.

[LO85] Lusk, E.L. ; Overbeek, R.A. :
Reasoning about Equality,
Journal of Automated Reasoning 1, 1985, pp. 209-228.

[Mc88] McCune, W.W. :
Otter 1.0 Users' Guide,
Report ANL-88-44, Argonne National Laboratory, Argonne, 1988.

[Po45] Polya, G. :
How to Solve It,
Princeton University Press, Princeton, 1945.

[Ru87] Rusinowitch, M. :
Demonstration automatique par des techniques de reecriture [in French],
These de Doctorat d'Etat en Mathematique, Nancy, 1987.

[SD81] Smith, R.G. ; Davis, R. :
Frameworks for Cooperation in Distributed Problem Solving,
IEEE Transactions on Systems, Man and Cybernetics, Vol. 11, No. 1, 1981, pp. 61-69.

[SLe90] Schumann, J. ; Letz, R. :
PARTHEO: A High-Performance Parallel Theorem Prover,
Proc. 10th CADE, LNAI 449, Springer, Kaiserslautern, 1990, pp. 40-56.

[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.

[Sm81] Smith, R.G. :
A Framework for Distributed Problem Solving,
UMI Research Press, Ann Arbor, 1981.

[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.

[Su89] Suttner, C.B. :
Learning Heuristics for Automated Theorem Proving,
M.S. Thesis, Technical University of Munich, 1989.

[Ul90] Ultsch, A. (ed.) :
Kopplung deklarativer und konnektionistischer Wissensrepräsentation,
Endbericht Projektgruppe PANDA, University of Dortmund, 1990.

[WRCS67] Wos, L. ; Robinson, G. ; Carson, D. ; Shalla, L. :
The Concept of Demodulation in Theorem Proving,
Journal of ACM 14, 1967, pp. 698-709.

[Ye86] Yeates, D. :
Systems Project Management,
PITMAN Publ., 1986.