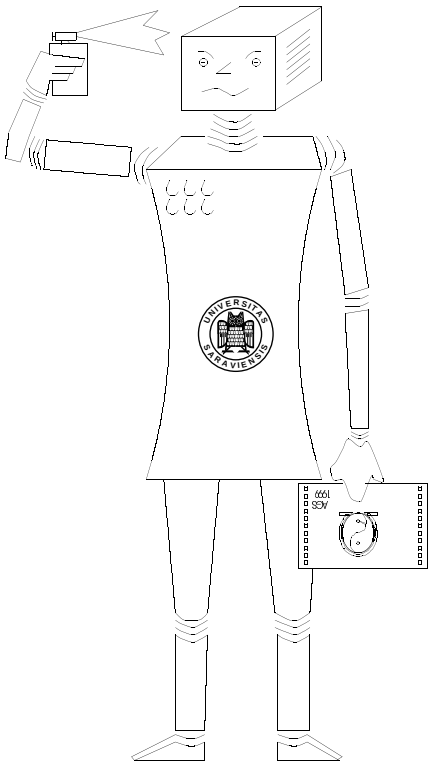
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/

SEKI Report
ISSN 1437-4447



Towards Concurrent Resource Managed Deduction

Christoph Benzmüller, Mateja Jamnik,
Manfred Kerber, and Volker Sorge

SEKI Report SR-99-07

Concurrent Resource Managed Theorem Proving*

Christoph Benzmüller[1], Mateja Jamnik[2], Manfred Kerber[2] and Volker Sorge[1]

[1]Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany
{chris|sorge}@ags.uni-sb.de
http://www.ags.uni-sb.de/

[2]School of Computer Science
The University of Birmingham, Birmingham B15 2TT, England
{M.Jamnik|M.Kerber}@cs.bham.ac.uk
http://www.cs.bham.ac.uk/

Abstract

In this paper, we describe an architecture for resource guided concurrent mechanised deduction which is motivated by some findings in cognitive science. Its benefits are illustrated by comparing it with traditional proof search techniques. In particular, we introduce the notion of focused search and show that a reasoning system can be built as the cooperative collection of concurrently acting specialised problem solvers. These reasoners typically perform well in a particular problem domain. The system architecture that we describe assesses the subgoals of a theorem and distributes them to the specialised solvers that look the most promising. Furthermore it allocates resources (above all computation time and memory) to the specialised reasoners. This technique is referred to as resource management. Each reasoner terminates its search for a solution of a given subgoal when the solution is found or when it runs out of its assigned resources. We argue that the effect of resource management leads to a less brittle search technique which has some advantages over the traditional search techniques such as breadth first or heuristic search. A prototype of the proposed system architecture is currently implemented in the ΩMEGA/MATHWEB framework.

1 Introduction

There are two major approaches to automated theorem proving, machine-oriented methods like the resolution method (with all its ramifications) and human-oriented methods. Most prominent amongst the human-oriented methods is the proof planning approach first introduced by [Bun88].

In this paper we argue that an integration of the two approaches and the simultaneous pursuit of different lines in a proof can be very beneficial. One way of integrating the approaches is to consider a reasoner as a collection of specialised problem solvers, in which machine-oriented methods and planning play different rôles. One of the main distinctions between machine-oriented and human-oriented methods is the generality of the approaches. While theorem provers that are built using machine-oriented methods are typically general purpose provers, human-oriented theorem provers make use of domain-specific knowledge.

Machine-oriented theorem provers like classical first-order theorem provers (e.g., BLIKSEM, OTTER, SPASS), analytical provers (e.g., SATCHMO or PROTEIN), and provers based on completion methods (e.g., EQP, WALDMEISTER) have reached a considerable reasoning power. This is

*This work was partly supported by EPSRC grant GR/M22031.

underlined by the recent solution of the Robbins problem by EQP [McC97]. However, these traditional systems follow fixed search strategies which are unlikely to fully model the problem solving expertise of human mathematicians. Furthermore, classical first-order theorem provers as well as higher-order theorem provers like TPS [ABI+96], or the LEO-system [BmK98] are often lost in the enormous search spaces stretched on a fine-grained calculus level. General complexity results demonstrate that no practical algorithm can be constructed which can solve arbitrary tasks. Even propositional logic is in a class that is generally considered intractable since it is NP-complete.

The success of human mathematicians can largely be ascribed to the fact that they are generally specialised in some fields and can rely on *domain-specific* problem solving techniques they accumulated throughout their professional experiences. Mathematicians learn during their academic training not only facts like definitions or theorems, but also problem-solving *know-how* for proving mathematical theorems [Ble86]. An important part of this know-how can be described in terms of reasoning methods like the diagonalisation procedure, the application of a definition, or the application of the homomorphy property. Human-oriented theorem proving tries to model this human approach by making use of domain-specific knowledge.

One approach to model human-oriented theorem proving on a computer is proof planning which adopts the planning paradigm. The so-called methods play the rôle of plan operators and their executions fill the gaps in a partial proof. Bundy views methods essentially as a triple consisting of a tactic, a precondition, and a postcondition. A tactic can be seen as a piece of program code that can manipulate the actual proof in a controlled way. A precondition and a postcondition form a declarative specification of the deductive ability of the tactic. The approach to mechanising reasoning using methods forms a qualitative step forward compared to a mere tactic language. Within such a planning framework it is now possible to develop proof plans with the help of the declarative knowledge in the preconditions and postconditions. In this view, proof planning makes use of traditional planning techniques in order to find proofs on an abstract level. Some state of the art proof planners are CLAM [BvHHS90], λ-CLAM [RSG98], and the proof planner of ΩMEGA [BmCF+97].

The concurrent resource managed theorem proving architecture described in this paper is developed within the framework of the ΩMEGA/MATHWEB architecture. An advantage of ΩMEGA is that it already provides various integrated classical reasoning systems (e.g., BLIKSEM, OTTER, EQP, SPASS, SATCHMO, PROTEIN, WALDMEISTER, TPS, LEO) as well as some specialised decision procedures (e.g., a constraint solver and the integrated computer algebra systems MAPLE [Red98] and $\mu\mathcal{CAS}$ [KKS98]), and an analogy module [Mel95]. Additional features are a multi-modal graphical user interface [SHB+99], a proof verbalisation tool [HF97] and a connected database of mathematical theories. Using the MATHWEB agent architecture [FHJ+99], most of these integrated systems can be distributed over the internet. Information on successful or unsuccessful proof attempts of the integrated systems (e.g., partial proofs) can be translated back into ΩMEGA's central proof data structure, which is based on a higher-order variant of Gentzen's natural deduction calculus. Translation of different results into the uniform representation in ΩMEGA clarifies the integrated results of very heterogeneous provers.

Human proof search behaviour may perhaps be best modelled as a mixture of proof planning, classical theorem proving, computing, and model generation. In ΩMEGA (and in related systems like ILF [Dah97] or DISCOUNT [DKS97], which integrate relatively homogeneous first-order reasoning systems) this largely has to be done by the user. Rather poor support is provided for a fruitful and guided cooperation of the available subsystems.

While ΩMEGA and MATHWEB provide the technical background, the work described here aims to investigate how we can establish a meaningful cooperation between different specialised problem solvers within a uniform architecture. We use as much as possible some of the already existing technology both in ΩMEGA and in other external systems (i.e., theorem provers, computer algebra systems, etc.). Therefore, the communication between specialised problem solvers is organised so that successful and unsuccessful proof attempts or partial proofs are communicated via ΩMEGA. The assessment of single subsystems and societies of subsystems is embedded within ΩMEGA as well as the shells surrounding the single subsystems in use. Hence, theorem provers which have communication features readily available (e.g., TPS) are used off the shelf, as black-box systems.

The information they provide is incorporated at run-time into the reasoning process searching for a proof of a conjecture.

The architecture that we describe in this paper uses resource management in order to determine the effort that is put into searching for a solution of a subgoal by each specialised reasoner. The overall result of this behaviour is a search technique that is less brittle than traditional heuristic search. There is less chance for a system to get lost in a misleading branch of a search tree. Hence, some of the problems that traditional heuristic search reasoners were unable to prove can now be automatically solved using our concurrent resource managed architecture. A comparison of this so-called focused search to breadth-first and heuristic search is given in 3.

The architecture we present here can be viewed as an agent architecture, where the specialised problems solvers can be considered as agents. The characteristics of an agent are that they are autonomous entities, which are aware of their own capabilities and those of other agents. They can communicate between each other, act on the environment, and have a learning capability in order to improve their own reasoning capabilities. The specialised subsystems in our architecture have some of these agent characteristics. Namely, like agents the solvers can run in parallel and act on the environment. The knowledge of an agent is initially provided by the user or the implementor of a single agent. Additional knowledge could be gained by evaluating successful and unsuccessful proof attempts in various mathematical domains as well as by feedback from other agents, i.e., solvers (for instance, the usefulness of results from some agents can be used in a reinforcement learning approach). However, our subsystems do not have learning capabilities yet. Moreover, they cannot directly communicate with other solvers and they do not have any knowledge of other systems. But they have a communication capability which is added to the solvers in our architecture in form of a shell built on top of a solver. The solvers communicate with each other via this shell and via a so-called planning cell manager (see 4.1).

We will refer to our subsystems (specialised solvers) as agents in this work for two reasons. First, our work is motivated by the work done in the multi-agent community, and second, we think that the view of the software components as agents with resources, tasks, and goals is a useful model for describing our approach. We are aware that currently the different components in our system have only some of the features which typically characterise agents. But we also believe that by enhancing these, we can model human theorem proving (and possibly other forms of reasoning as well) more adequately.

2 Motivation

Here we motivate our work on emulating human mathematical reasoning as an interactive process between a number of different problem solving strategies within a heterogeneous theorem proving framework, which makes use of proof planning and traditional machine-oriented theorem proving. We first give some speculative evidence supported by advice that other scientists have given about the importance of following different solving strategies within any single attempt to solve a problem. Second, we discuss the relationship between reactive and deliberative reasoning processes and suggest that both should be integrated into a cognitively adequate system. Third, we introduce the technique of resource managed and concurrent reasoning which is employed in our system. Finally, we list the type of problems that we aim to solve by our system, and which have not been solved automatically or in a general framework before.

2.1 Modelling Human Reasoning

Human reasoning has been described in traditional AI (e.g., expert systems) as a process of applying rules to a working memory of facts in a recognise-act cycle. In each cycle one applicable rule is selected and applied. While this is a successful and appropriate approximation for many tasks (in particular for well understood domains), it seems to have some limitations, which can be better captured by an approach that is not only cooperative but also concurrent.

Many of us have experienced that collaboration can lead to results that each single person could not have achieved. One of the rare documents in the history of mathematics where such a successful collaboration and the joint development of ideas is described in detail is provided by van der Waerden in [Wae64]. The document shows how three persons contributed to a proof by different ideas, not only by cooperating but also by concurrent search. It may seem to be too ambitious to model the cooperation of different persons at a time when we have not yet developed an adequate model for the mathematical reasoning of a single person. Minsky [Min85], however, gives convincing arguments that the mind of a single person can and should be considered as a society of agents. Put in the context of mathematical reasoning this indicates that it is necessary to go beyond the traditional picture of a single reasoner acting on a working memory – even for adequately describing the reasoning process of a single human mathematician.

In the following we want to support Minsky's argument by linking it to a key work in the study of mathematical reasoning, Hadamard's "Psychology of Invention" [Had44]. In his study Hadamard describes the predominant rôle of the unconsciousness when humans try to solve hard mathematical problems. He explains this phenomenon by its most important feature, namely that it can make (and indeed makes) use of concurrent search (whereas conscious thought cannot be concurrent), see [Had44, p. 22]:

> *"Therefore, we see that the unconscious has the important property of being manifold; several and probably many things can and do occur in it simultaneously. This contrasts with the conscious ego which is unique. We also see that this multiplicity of the unconscious enables it to carry out a work of synthesis."*

That is, in Hadamard's view, it is important to follow different lines of reasoning simultaneously in order to come to a successful synthesis. Some pages later, on p. 46 he describes a view of Poincaré, namely to compare ideas to atoms — only when they meet they can react, simultaneous movement of many atoms enlarges their chances to meet:

> *"Again comparing ideas to Poincaré's atoms, it may happen that the mind projects them, exactly or almost exactly, in certain determinate directions. Doing so has the advantage that the proportion of useful meetings between them happens to be relatively great compared to the sterile ones ... This is what Souriau expresses by the quite striking phrase: "In order to invent, one must think aside" ... we can remember Claude Bernard's statement, "Those who have an excessive faith in their ideas are not well fitted to make discoveries."* and on p. 54: *But this [thinking aside] is not yet completely satisfactory: in this way we shall think of expected directions for "aside" thought ..."*

To rephrase it, Hadamard states that concurrent search allows for combinations of ideas, which are projected either exactly or almost exactly in a particular direction. Inventions require a lot of knowledge, but they need a lot of (informed) search as well. For difficult proofs, it is not sufficient to follow a standard approach, but it is necessary to "think aside", that is, to follow paths in the search space, which are not as promising. But Hadamard also stresses that just thinking aside – in the terminology of search, just to search somewhere – is not enough in order to be successful. We must limit the search to promising directions. In other words, while best-first search may be trapped in just following the standard approach and may not work, the search spaces are far to big in order for breadth-first search to work.

2.2 Deliberation versus Reactiveness

A classical approach to model intelligence consists of carefully selecting a knowledge representation formalism and then modelling parts of a domain in this formalism. As an antithesis an approach has been developed that explicitly does not make use of knowledge representation and complicated deliberations – Brooks phrased it as "*Intelligence without Reason*" [Bro91]. In this approach it is possible to obtain complex, apparently goal directed and intentional behaviour which has no

long-term internal state and no internal communication. This is referred to as a reactive form of behaviour modelling. For a detailed discussion of Brooks' approach and its relationship to theorem proving see [Bun94].

Recent years have seen an attempt to reconcile the deliberative and the reactive approaches in single agent architectures [Slo99]. This is partly motivated by looking at the human way of acting and reasoning which can be better explained as a combination of the two cases rather than by any one of them alone. Also, practical issues play an important rôle: in certain cases reactive behaviour is computationally more efficient, while in others reactive behaviour gets stuck. In the latter case deliberative behaviour can sometimes prevent blocking of a reasoning process.

2.3 Concurrency and Resource Management

A weakness of most state of the art reasoning systems is that they usually follow rigid and inflexible solution strategies in their search for proofs. Instead, human mathematicians use — depending on their level of expertise — "*a colourful mixture of proof techniques*" (as Wittgenstein phrases it). In an attempt to prove a mathematical theorem they typically first try a well known standard technique in the focus of the mathematical theory. If this technique does not lead to the wanted results in a reasonable amount of time, they may doubt that the theorem holds at all and look for a counterexample. If this also fails, they may try again by widening and/or deepening the proof search.

The aim of our approach is to emulate this flexible problem solving behaviour of human mathematicians in a concurrent resource management based reasoning approach. Thus, our system reflects at least some of the ideas of a *sophisticated and experienced problem solver* as described by Pólya in [Pól65, I, p. 64]:

> "*... when he does not succeed in guessing the whole answer, [he] tries to guess some part of the answer, some feature of the solution, some approach to the solution, or some feature of an approach to the solution. Then he seeks to expand his guess, and so he seeks to adapt his guess to the best information he can get at the moment.*"

In [Pól65, II, p. 93] Pólya describes rules of discovery which can be supported by a concurrent search effort. By these rules, more resources are spent on the more promising, while some (but less) resources are spent on the less promising search directions:

> "*The less difficult precedes the more difficult. ... The more familiar precedes the less familiar. An item having more points in common with the problem precedes an item having less such points. ... Less remote parts precede more remote parts.*"

2.4 Examples for Multiple Problem Solvers

With our approach we want to tackle two kinds of mathematical problems, firstly, problems that are currently not automatically solvable by any system, and secondly, problems that can currently only be solved by very specialised systems. While the motivation for the first class of problems is self-evident, the latter problems are interesting as well: on the one hand in order to answer the question whether a general mathematical reasoner can adequately deal with such problems, and on the other hand it addresses an important aspect of user friendliness of a system. Users should be able to perform various reasoning tasks within the same system. They should be able to formulate their problems in a uniform and elegant way, without the need to know every specialised system that might be appropriate for the task at hand.

We demonstrate our approach by three examples which we briefly introduce here and which we discuss in more detail in 6.

Our first example is a theorem where the equality of $\log(x^n)$ and $n \cdot \log(x)$ is derived from the fact that two functions are equal if and only if their derivatives are equal and they have one common value.

$$\text{(I)} \qquad \begin{array}{c}[\forall f_{\blacksquare}\forall g_{\blacksquare}(f' = g' \wedge \exists x_{\blacksquare} f(x) = g(x)) \Leftrightarrow f = g] \\ \Rightarrow \forall n_{\blacksquare}(\lambda x_{\blacksquare} n \cdot \log(x) = \lambda x_{\blacksquare} \log(x^n))\end{array}$$

Here the notions of a derivative (f' is the derivative of f and so on) and of the logarithm need to be defined in a database which is available to the system. The proof of theorem (I) involves both reasoning and calculation. Although the computation tasks like computing the derivative or calculating $n \cdot \log(x) = \log(x^n)$ for a particular value could still be done by a deduction system (if an appropriate definition of the log function is available), these computations are tasks which can typically be more efficiently solved by some computer algebra system.

Our second example demonstrates how specialised automated reasoners and a proof planner can fruitfully cooperate. In theorem (II) a is a binary relation, $\circ$ denotes composition of binary relations (we assume that $\circ$ is defined for all binary relations u, v and all elements x, y as $u \circ v(x, y) \Leftrightarrow \exists z_{\blacksquare} u(x, z) \wedge v(z, y)$)) and a^n is n times the composition of a.

$$\text{(II)} \qquad \forall n_{\blacksquare}\forall a_{\blacksquare}(a^n \circ a) = (a \circ a^n)$$

Theorem (II) can be proved by a first-order induction theorem prover given that the user encodes the problem appropriately. However, in our approach, the straightforward higher-order formulation leads to an efficient proof built on the cooperation of a theorem prover for extensional higher-order logic and standard first-order logic. The possible transformations between the encodings are done by the cooperating reasoning systems in the background, unnoticed by the user.

Our third example stems from the TPS-library and has already been discussed in detail in [BBS99]. It states that if there is a partition p of some set, then there is an equivalence relation q whose equivalence classes are exactly the elements of p:

$$\text{(III)} \qquad \forall p_{\blacksquare} \mathit{partition}(p) \Rightarrow (\exists q_{\blacksquare} \mathit{equivalence\text{-}rel}(q) \wedge (\mathit{equivalence\text{-}classes}(q) = p))$$

Note that *partition*, *equivalence-classes*, and *equivalence-rel* are derived higher-order concepts that need to be explicitly formalised in some database available to the system. The proof of theorem (III) requires a fair amount of higher-order equality and extensionality reasoning which cannot be done by a single higher-order automated theorem prover so far (or any other system we know of).

We will discuss how proofs for the first two problems can be derived in our system architecture in more detail in 6. The third example is currently being studied in detail, and the search for its proof is not fully automated yet. However, we identified all the tasks that need to be done within our architecture in order to automate the search for the solution of the third example. In 6, we will concentrate especially on how different systems have to cooperate in order to contribute to the solutions of the given problems.

3 Search Strategies and Brittleness

Search is a major feature of most AI-systems, as a consequence search techniques are well investigated and descriptions can be found in standard textbooks in the field like [RN95] and in specialised textbooks like [Pea84]. In this section we discuss some search techniques which are relevant to our approach. Breadth-first search is a technique which in theory guarantees to find a solution to a problem, if there is one. The downside of this technique is that it is computationally very expensive and in fact turns out to be unusable in practice. Hence, heuristic search was invented which uses heuristic knowledge built into the system in order to prune the search space. This technique proves to be very powerful in practice. Its main drawback is that in many application areas no really good heuristics are known. In particular, heuristics are normally not complete, that is, they do not guarantee to find an existing solution. It is possible that the heuristic may guide the search down a branch of the search tree which does not lead to a solution, so the search may get lost and no solution is found at all.

The resource management technique described in this paper controls the amount of effort put into solving a problem by each individual subsystem. The periodic evaluation and assessment of the success of each reasoner insures that the search of that particular reasoner can rarely get lost in a branch of a search tree that does not lead to a solution. The overall effect of controlling concurrently the resources of a number of reasoners, and evaluating them periodically, is that the search of the top level system is less brittle, and more distributed than heuristic search. Yet, unlike breadth-first search, it is in many cases still practically feasible. We call this search a focused search technique.

Figure 1 demonstrates the differences between breadth-first, heuristic and focused search tech-

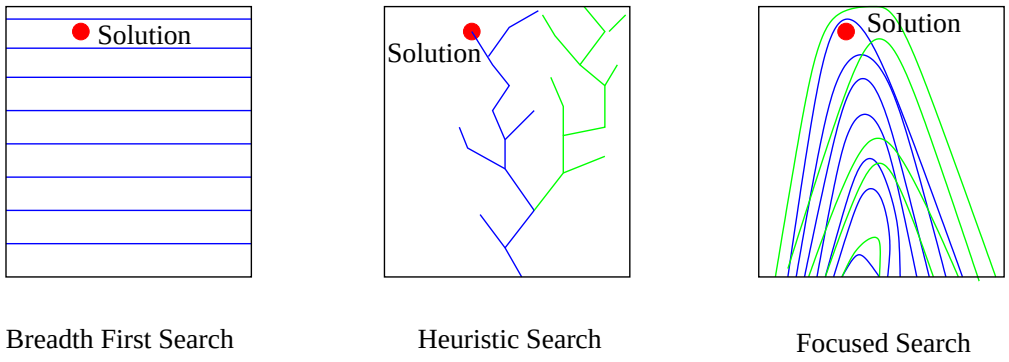


Figure 1: Focused search in relation to the breadth-first and heuristic search techniques.

niques. Breadth-first search is robust in the sense that it is impossible to miss a solution. However, it is normally prohibitively expensive. Heuristic search [1] may be considered as the other extreme case, it is possible to go with modest resources very deep in a search tree. However, the search is brittle in that a single wrong decision may make it go astray and miss a solution, independently of how big the allocated resources are. Focused search can be considered as a compromise — it requires more resources than heuristic search, but not as much as breadth-first search. As a result, a solution can still be found even if the focus of the search is misplaced. Clearly, more resources are necessary in the case of a bad than of a good focus.

Figure 2 demonstrates the comparison between the shapes of search spaces for different search techniques, given a certain amount R of resources. Breadth-first search looks across all the

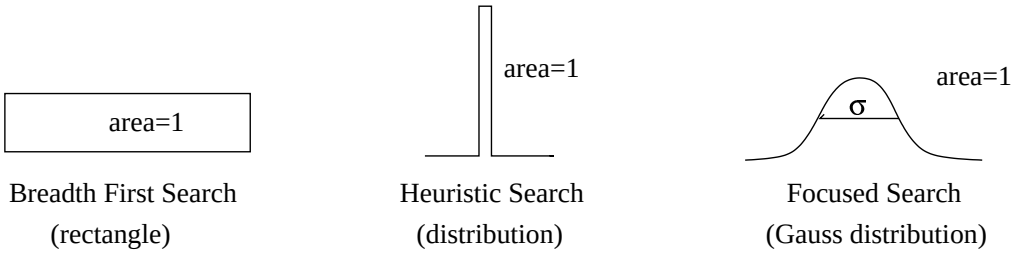


Figure 2: Shapes of search spaces for a given amount of resources R for breadth-first, heuristic and focused search techniques.

branches of a search tree at any one time. The shape of its search space for a given amount of resources can be thought of as rectangular. Heuristic search looks only down some branches of a search tree which are suggested by the built-in heuristics. Hence, the search may be brittle.

[1] Of course, this description is very simplified. For the illustration purposes we actually use only one kind of heuristic search, namely best-first search.

The shape of its search space for a given amount of resources looks like a steep distribution curve. Focused search explores the general area that particular specialised heuristics of a number of reasoners suggest. Hence, it is more spread out and less brittle than heuristic search. The shape of its search space for a given amount of resources can be approximated by a Gaussian distribution curve.

Depending on the variance σ^2 it is possible to simulate breadth-first search (no brittleness, low search depth for a fixed amount of resources) and heuristic search (high brittleness, high search depth for a fixed amount of resources – a good heuristic is assumed) as extreme cases. The more you focus the more brittle is the search, the less you focus the more robust it is. So the concurrent resource-bounded search subsumes different extreme cases and can be triggered to model the intermediate cases as well. The invariance σ^2 is changed by changing the amount of resources allocated to the specialised provers.

In practice, the focused search technique can be more successful in finding a solution to a problem than the traditional heuristic search technique. Later in the paper we will show an example of a theorem which goes beyond the strength of existing theorem provers, but which could be solved in our resource managed system. More research is necessary in particular in form of empirical studies to compare heuristic and focused search. In particular, it would be interesting to know whether a monotonicity assumption can be made for existing automated theorem provers. That is, if a reasoner is given a certain amount of resources and it succeeds to find a solution, would it still be able to find this solution if it is given more resources? If it does, and therefore the reasoner is monotonic, then a monotonic resource allocation to all of the monotonic reasoners in our architecture would preserve the monotonicity property of the overall system.

4 Architecture

The architecture that we describe here allows a number of proof search attempts to be executed in parallel. Each specialised subsystem may try a different proof strategy to find the proof of a conjecture. Hence, a number of different proof strategies are used at the same time in the proof search. However, following all the available strategies simultaneously would quickly consume the available system resources consisting of computation time and memory space. In order to prevent this, and furthermore, to guide the proof search we developed and employ a resource management concept in proof search. Resource management is a technique which distributes the available resources amongst the available subsystems (cf. [Zil95]). Periodically, it assesses the state of the proof search process, evaluates the progress, chooses a promising direction for further search and redistributes the available resources accordingly. If the current search direction becomes increasingly less promising then backtracking to the previous points in the search space is possible. Hence, only successful or promising proof attempts are allowed to continue searching for a proof. This process is repeated until a proof is found, or some other terminating condition is reached. An important aspect of our architecture is that in each evaluation phase the global proof state is updated, that is, promising partial proofs and especially solved subproblems are reported to a special plan server that maintains the progress of the overall proof search attempt. Furthermore, interesting results may be communicated between the subsystems (for instance, an open subproblem may be passed to a theorem prover that seems to be more appropriate). This communication is supported by the shells implemented around the specialised problem solvers. The resource management mechanism analyses the theorem and decides which subsystems, i.e., which provers, should be launched and what proportion of the resources needs to be assigned to a particular prover. The mechanism is also responsible for restricting the amount of information exchange between subsystems, so that not all of the resources are allocated to the communication. Figure 3 demonstrates this concurrent resource management based proof planning architecture.

Clearly, the evaluation of the success of a proof strategy is crucial for determining the amount of resources that is allocated to a subsystem. This evaluation is based on the contribution that the subsystem has made in the proof attempt as well as on its prospect of success in the rest of the search. For example, a favourable contribution is a partial problem solution. The future

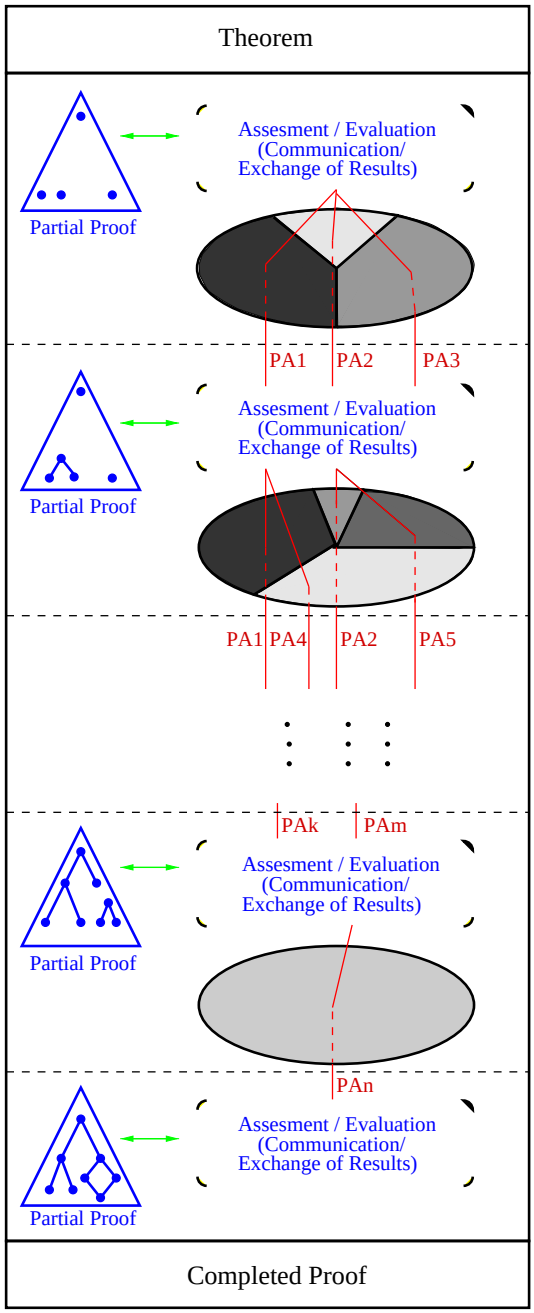


Figure 3: The reasoning process – iterative allocation of resources to specialised provers (PAx) by assessment/evaluation, and the subsequent construction of a proof of a given theorem.

prospect of a specialised problem solver is estimated with respect to the updated global proof tree and according to the information communicated between the subsystems.

4.1 System Components

We realise the so-called focused proof search as an adaptation of the multi-agent planning architecture, MPA, [WM98] in the proof planning domain. The main component of MPA is a multi-agent

proof planning cell, which consists of several planning agents, a plan server, a domain server, and finally a planning cell manager. The comparison between MPA in [WM98] and our architecture is given in 7.

4.1.1 Planning Agents

To build up a concrete multi-agent proof planning cell we choose a finite subset of agents $\mathcal{A} := \{A_1, \ldots, A_n\}$ from a given pool of planning agents. A planning agent either consists of a single reasoning system or of a composition of reasoning systems. Generally, a planning agent encapsulates any reasoning system (or a composition of systems) that is capable of reading and returning partial proof plans in a uniform representation format. Figure 4 illustrates graphically the structure of

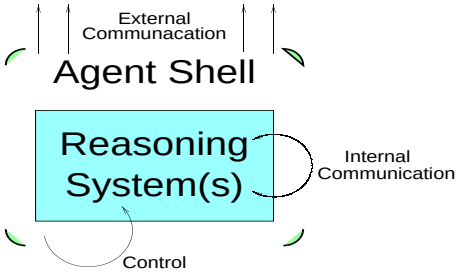


Figure 4: Structure of a planning agent.

a planning agent. In general, a planning agent may even encapsulate another multi-agent proof planner, that is, we allow our architecture to be recursive. Consequently there may exist internal communication among the encapsulated subsystems. The task of the agent shell is to provide a standardised communication protocol between agents and the planning cell manager as well as amongst the planning agents themselves, and to control the encapsulated systems, for instance, with respect to resource handling. However, the pool of proof planning agents suggested below consists only of single reasoning systems or rather simply structured compositions of reasoning systems.

4.1.2 Plan Server

A plan server stores partial proof plans returned by the planning agents in their previous runs within the unified data format. In addition to the currently focused partial proof plan it also stores alternative partial proof plans delivered by the planning agents so far. This enables backtracking on two distinct levels: on the one hand, we can backtrack within the actual proof plan by taking back single proof steps or subproofs contributed by some of the planning agents. On the other hand, we can completely shift to some alternative proof attempt that has been abandoned previously.

In addition, the plan server provides a communication blackboard that is associated with the currently focused partial proof plan. This blackboard provides communication facilities for the running agents. For example, a higher-order reasoner may want to pass the generated first-order clauses to the centre of expertise for first-order logic to be checked for inconsistency (for details see 6).

4.1.3 Domain Server

A domain server provides the necessary knowledge for the planning cell manager as well as for the single planning agents. In our context it consists of a structured database of mathematical theories. Each theory contains definitions of mathematical concepts, related lemmata and theorems together with their proofs and possibly a collection of open problems. Moreover, it contains domain specific knowledge relevant to certain planning agents.

Additional information stored by the domain server includes statistical information about which subsystem proved which theorem, how fast the subsystem proved the theorem, in which mode did the subsystem prove the theorem, etc. Failed proof attempts can also be associated with the open problems in the domain server. All this information is useful for the assessment of the suitability of a subsystem to solve a problem, and the subsequent distribution of the resources to this subsystem in a proof search attempt.

For example, let us assume that a subproblem in the focus belongs to a certain class of theorems in some domain. If the information stored in the domain server suggests that a particular subsystem was successful in solving theorems from this domain in the past, then this subsystem should be preferred and get the largest amount of resources allocated in the subsequent proof search attempt.

4.1.4 Planning Cell Manager

Tasks of the planning cell manager are to evaluate the resulting partial (or complete) proof plans delivered by the planning agents at the end of each running phase, and to update the plan server accordingly. This means that the most promising partial proof plans are selected and inserted into the backtracking tree according to their particular evaluation result. The non-interesting partial proof plans are therefore filtered out. This generally leads to a new partial proof plan to be in the focus of the next planning phase. Once a new partial proof plan has been selected, the planning cell manager has to allocate new resources to the available planning agents for the next planning phase. This focuses the proof search on a particular direction based on the examination of the selected partial proof plan.

4.2 The Running System

We already illustrated in Figure 3 that our system iterates single reasoning phases in which the planning agents concurrently try to solve the problem in the focus. We furthermore hinted that each phase is realised by three subsequent steps: assessment and distribution of resources, concurrent reasoning, and evaluation of results. These steps will now be discussed in more detail.

4.2.1 Assessment and distribution of resources

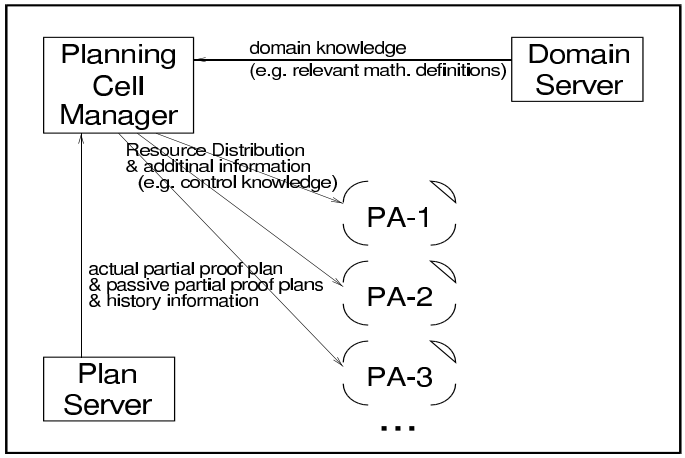


Figure 5: Assessment and distribution of resources.

The task of this step is to compute an appropriate resource distribution for the planning agents used in the next run. As a result of this step we obtain a triple (t, m, h) for each planning agent in the planning cell, where t specifies a time-bound, m a memory-bound and h a particular host

system for executing the computations. Planning agents can be excluded from search by setting their time-bound (or memory-bound) to 0.

In our prototype implementation we aim to realise a restricted version of the general architecture described above, namely a system with synchronised runs. Thus, the agents either obtain the time-bound 0 (i.e., they are excluded from the next run) or a time-bound a that is fixed for all agents in the next run. One can give preference to the most promising agents by assigning them to the most powerful host systems available.

Furthermore, in our prototype, the evaluation and resource distribution is still executed mainly by the user. One of the most challenging research tasks is to develop more assessment criteria appropriate for automating the evaluation. However, the goal is not to remove the human from this process. On the contrary, our approach strongly facilitates the communication between the human and the machine. In fact, human expertise is incorporated during the assessment phase. Therefore, it is important to provide an elaborate user-interface for supporting human-machine interaction during the assessment and distribution task.

Some assessment criteria suitable for automation are:

(i). *The logic that the current problem and its proof belong to:* for example, in case of a first-order problem, higher-order reasoners should obviously be excluded from the search. A proof planner should probably be invoked in a special first-order reasoning mode (if available).

(ii). *The mathematical theory that the current problem and its proof belong to:* for example, if a problem belongs to the limit domain, then the assessment module may give preference to the ΩMEGA-proof planner, which is currently the only available system with strengths in this domain. The knowledge about the strengths in particular mathematical domains for particular systems can be received from the domain server.

(iii). *The similarity of a current subproblem to some already solved theorems stored in the domain server:*[2] if similar problems are found, preference is given to the planning agent that solved these before. Furthermore, if available, preference is given to a planning agent which uses analogy reasoning techniques.

4.2.2 Concurrent reasoning

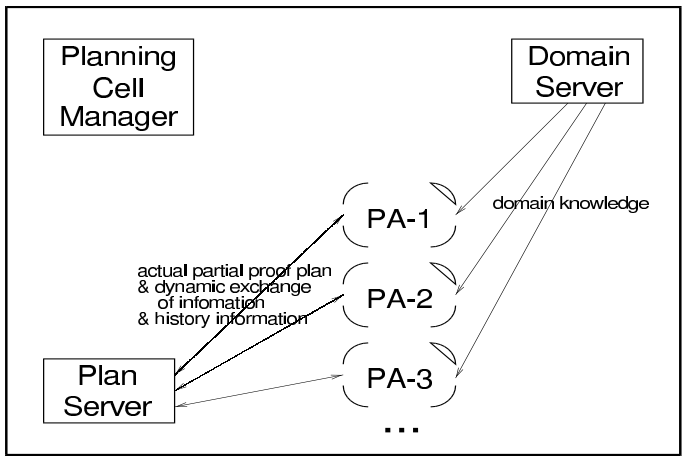


Figure 6: Concurrent reasoning.

According to the resource distribution computed in the assessment step the planning agents then start to reason about the problem in focus. The problem in focus can be requested by the

[2]For this task we employ the similarity checker of ΩMEGA's analogy component [Ull00]. In this context the analogy planning agent (as sketched in 5.1) is important. It is is capable of reformulating ΩMEGA-proofs.

agents from the plan server. The agents quit their search either when their computations terminate or as soon as their resources are consumed. In the former case agents can return either some useful result or failure, while the latter case will be recorded as failure automatically if the systems are not able to return some useful results. A useful result can be of one of the following types: an indication of the successful solution of a problem, a result of a computation or a completed or partial proof plan.

Interesting data can be exchanged between the concurrently running planning agents via a special communication blackboard maintained by the plan server. For instance, a planning agent for higher-order resolution reasoning may pass pure first-order clauses generated within its reasoning process to the planning agent for first-order reasoning. This first-order reasoner then checks whether the generated first-order clauses are unsatisfiable (see 6 for further details). In our prototype the run-time communication via the communication blackboard should be reduced as much as possible in order to avoid a communication bottleneck. Although we support the communication between higher-order reasoners and first-order reasoners via this channel, we do not want to establish a communication between specialised automated first-order reasoning systems as for instance, suggested in [DF99]. In our architecture, a centre of expertise for first-order reasoning should be encapsulated itself as a planning agent (see also the description of the first-order planning agent sketched in 5.1).

4.2.3 Evaluation of results

After terminating their computations the agents return completed or partial proof plans in the uniform data format to the planning cell manager. The cell manager then evaluates the incoming

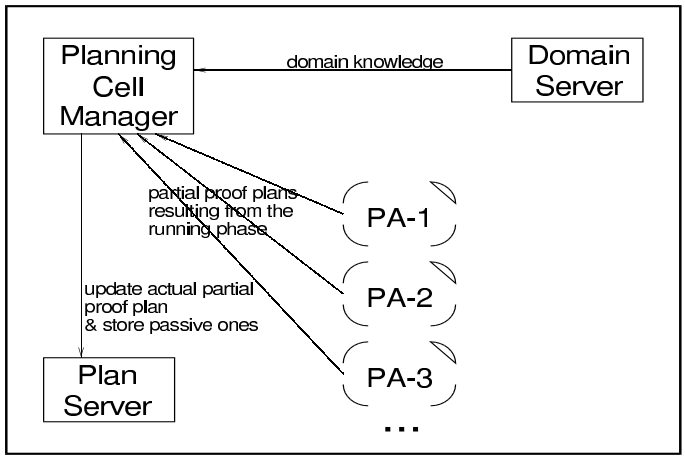


Figure 7: Evaluation of results.

partial proof plans and encodes the results of this evaluation process, for instance, in form of an integer value assigned to each partial proof. This integer value indicates how promising a particular proof plan is for further investigation. After filtering out all non-promising results, the remaining partial proof plans are reported to the plan server. The plan server stores them in a preference list according to their evaluation results. The best rated partial proof plan becomes the current proof plan in the next run. If the evaluation of the partial proof plans indicates that the system searched in a non-promising direction, then the system can use the preference list of previously evaluated proof attempts in order to continue its search in a more promising direction.

As mentioned in the assessment and resource distribution step, one of the most challenging research tasks is to automate the evaluation process. Again, we do not want to exclude the user from this crucial task. On the contrary, we support the user by providing an elaborate user-interface that offers appropriate evaluation criteria which are suitable for automation.

The evaluation criteria are:

(i). *The selected partial proof plan should be new with respect to the proof plans previously processed by the plan server.*

(ii). *There are more simplified expressions in the partial proof plan than before.*

(iii). *The open goals in the selected partial proof plan must be simpler.* By simpler we mean, for example, an open goal has been reduced from higher-order to first-order logic, or from first-order to propositional logic.

(iv). *The open goals are similar to previously proved theorems, which are stored in the domain server.*

5 Implementation

In this section we detail which part of the architecture presented is currently implemented in a first prototype. Figure 8 illustrates the design of this prototype which is only partially completed so far. Moreover, some of the subsystems employed in our architecture still require modifications.

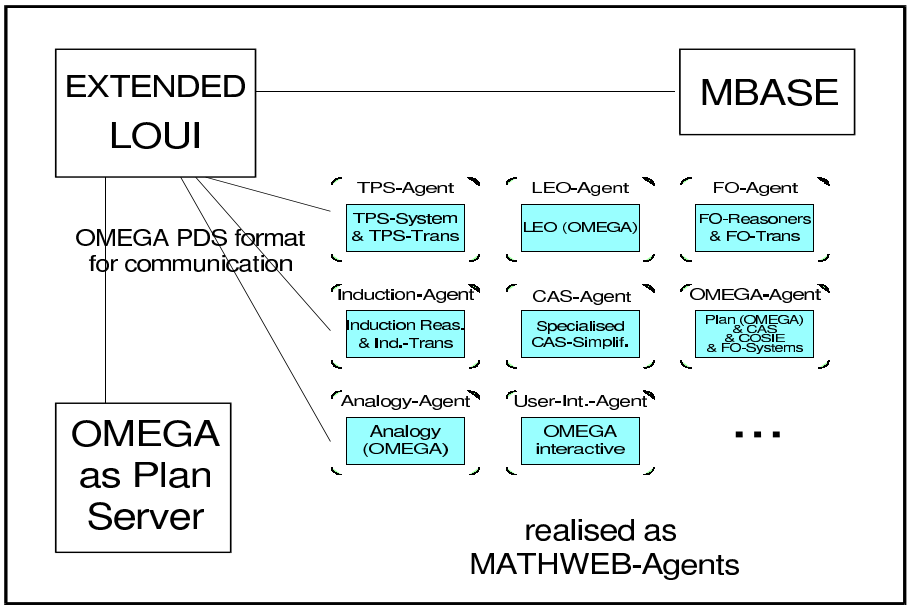


Figure 8: Implementation and design decisions: concrete choices of subsystems.

In the prototype implementation we tried to draw as much as possible on the already existing infrastructure in order to evaluate the feasibility of the proposed architecture. Therefore, we used the core of the ΩMEGA-system as starting point of the implementation.

As a first approximation to the plan server we use ΩMEGA's partial proof plan data structure (PDS) which maintains ΩMEGA-style generalised natural deduction proofs simultaneously on different layers of abstraction [CS99]. The contributions of most of the integrated agents can be incorporated or translated into the PDS format. For instance, terms computed by the computer algebra system can be translated into ΩMEGA's syntax and incorporated into the proof, or refutation proofs by first-order theorem provers can be translated into PDS-style natural deduction proofs. Thus, the PDS in our prototype serves as a uniform communication data structure. In the future, we aim to adapt it and develop a PDS format which is less complex, optimised and more pertinent to our setting. This is currently being developed.

The main research task is the development of an intelligent planning cell manager. Currently we have expanded ΩMEGA's mechanism to suggest commands for interactive theorem proving [BmS98]. This mechanism is based on a multi-agent architecture where the single agents test

rules and tactics for the applicability in a given proof state. They then present the applicable ones to the user in a heuristically ordered way. The suggestion mechanism provides resource adaptive behaviour for single command suggestion agents. Moreover, it can classify certain subgoals as appropriate for a specific automated theorem prover, and it can suggest its use to the user [BmS99]. Like ΩMEGA's core system, this whole mechanism is implemented in LISP.

The classification agent also realises the first two assessment criteria given in 4. The preference for calls to a specific system is enforced using the heuristic sorting functions for the suggestion mechanism. These are presented to the user via a graphical display of the blackboard containing all sorted suggestions. Moreover, we have automated the process of assessing and executing to an extend that the suggestion mechanism carries out a certain predefined number of steps automatically. The backtracking for this automation is, however, still rudimentary. It works only for backtracking single step tactic applications, but not yet for the whole subproofs inserted by some reasoning component.

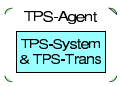
Calls to system components, other than the simple application of tactics, are carried out in the background. For this, in particular for calls to MAPLE, we partially employ the MATHWEB architecture [FHJ+99] into which ΩMEGA is embedded. MATHWEB has an architecture that is similar to CORBA (Common Object Request Broker Architecture) [Sie96]. Systems are integrated as mathematical services by embedding them in agent shells. The shell enables the communication with other services via the domain-independent general communication language KQML [FF94]. MATHWEB can automatically distribute the agent shells to different host systems over the internet, and manage the maintenance of processes and the message handling. However, the resource concept of MATHWEB is not yet fully developed. This forces us to restrict ourselves, for now, to allocating computation time as resources only to the single agents.

The contributions of automatic components in the form of suggestions for further proof attempts can be displayed on the blackboard. However, due to only partial implementation of a plan server, the choices of subproofs can be displayed only one at a time, rather than several choices at the same time. The implementation of the display of several choices of the subproofs on the blackboard simultaneously will require to keep several copies of a PDS in parallel, and an extension of the graphical user interface LOUI [SHB+99]. Presently, the user has to make an appropriate choice.

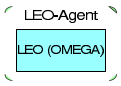
Finally, we employ ΩMEGA's knowledge base which consists of a hierarchically structured collection of theories as our domain server. The theories contain basic definitions, axioms, theorems and tactics for particular mathematical domains as well as domain specific knowledge for ΩMEGA's proof planner. As displayed in Figure 8, we aim to employ the MBASE-system [FK99] as domain server. However, MBASE is still in the early development phase and thus, is not yet suitable for our purposes. Once it is completed, it will not only contain mathematical knowledge in different representations, but it will also be accessible from different types of mathematical reasoning systems. Moreover, it is planned to have MBASE as an independent system within MATHWEB. This will make our architecture more independent from the ΩMEGA core system.

5.1 Selected Planning Agents

Here, we describe a few specialised reasoning agents used in our architecture.



The TPS-agent provides the TPS-system [ABI+96] – an automated reasoning system for higher-order logic — together with a special module of the ΩMEGA-system that is capable of translating partial proofs from the PDS format into corresponding partial proofs in the input format for TPS and vice versa. During the transformation process the translation module looks up in the domain server the unexpanded mathematical definitions given in PDS format (see 4.1), and makes this information available to the TPS-system. Hence, TPS can use its mechanism for selectively expanding definitions [BA98]. The result of this agent, after consuming the available resources, is either a completed or a modified partial proof.



The LEO-agent encapsulates the resolution based higher-order theorem proving system LEO [Ben99] which specialises in simple extensionality reasoning. The LEO-system is implemented as part of the ΩMEGA-system. Hence, this agent provides a particular instance of the ΩMEGA core system. However, our system employs only the features belonging to the LEO-subsystem. LEO often generates a set of contradictory first-order clauses when using extensionality reasoning to solve problems [Ben99]. Since LEO lacks facilities for efficient first-order reasoning it can take a while for it to realise that the clauses are contradictory, or it might even fail to do so altogether. Therefore, it is advisable to pass first-order clauses generated by LEO to specialised first-order reasoners in order to improve the overall capability of the system. This idea is illustrated in detail in 6.

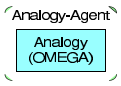
The FO-agent is a collection of the state of the art first-order reasoners (OTTER, SPASS, BLIKSEM, and PROTEIN), as well as special first-order equational reasoners (Waldmeister and EQP). In our prototype implementation the FO-agent runs the embedded reasoners in parallel. It uses some knowledge of the structure of the proof problem and the expertise of the embedded systems by being able to exclude the equational reasoners from proving non-equational problems. Results from the first-order provers can be integrated into the overall proof by employing a translation module that can translate refutation proofs into PDS-style natural deduction proofs [Hua94, Mei97]. This module is an integral part of the ΩMEGA-system, but is currently being made available as a stand-alone component in the MATHWEB environment. In case several of the first-order reasoners return a proof within their specified resource limitation, then the proof which translates into the shortest natural deduction proof is preferred.

So far, we do not yet organise any further communication between the single first-order provers. However, we intend to replace this rather simple agent by a more elaborate centre of expertise for first-order reasoning. For instance, one could integrate or rebuild a system as sketched in [DF99], enhance it with an appropriate translation module, and encapsulate it into an agent shell.

The CAS-agent employs computer algebra systems in order to simplify the current proof problem. It is a wrapper for SAPPER, a generic interface that can connect arbitrary computer algebra systems to ΩMEGA [KKS98]. SAPPER integrates the general purpose computer algebra systems MAPLE, GAP and Magma (two systems specialised on group theory), and our self-tailored $\mu\mathcal{CAS}$ system (a small computer algebra system specialised on polynomial manipulations and arithmetic) which has the ability to generate proof plans for its computations. Since SAPPER also realises the syntax translation for the single system, the CAS-agent can use this knowledge to decide whether any expression within one of the proof problems can be passed to one of the integrated computer algebra systems. It then passes these expressions to the appropriate computer algebra system — possibly to several in parallel — and, in case the expression was successfully simplified, returns the resulting simpler proof problem. Thus, computer algebra systems are currently used as black boxes, except in the case of the $\mu\mathcal{CAS}$ system which also returns partial proof plans in PDS format.

The ΩMEGA-agent provides a particular instance of the traditional ΩMEGA-proof planner together with the employed external subsystems. The state of the art ΩMEGA proof planner employs the COSIE constraint solver, the computer algebra system MAPLE, and several first-order reasoners. As ΩMEGA operates directly on partial proofs in the PDS format, this agent after consuming the available resources, returns as a result either a complete or a partial proof.

The automated agents and the user may work concurrently on the problem in focus. The interactive user session is made available by an additional planning agent which provides (another) instance of the ΩMEGA-system together with the user interface LOUI for interactive proof development. Note that our system behaves like a traditional interactive theorem proving environment in case the planning cell manager gives preference to the results of the user in all evaluation phases.



The analogy agent encapsulates the analogy reasoning system described in [Ull00]. This system looks for target theorems in the domain server that share major similarities to the given source problem. If it finds a suitable target theorem it then tries to use the target proof to solve the source problem. The result of the reformulation process is either a complete or partial proof in PDS format.

So far, we do not have any reasoning component specialised on induction integrated into the system. However, the INKA-system [AHMS99] is currently being integrated into the MATHWEB environment. Thus, we will be able to employ this system soon. Since INKA uses a sequent calculus and stores proofs in a PDS-like structure, it will be straightforward to integrate the subproofs found by INKA with the help of an appropriate translation module. In the long run, it would be desirable to integrate several induction reasoners in order to form a respective centre of expertise. Hence systems such as CLAM [BvHHS90] and λ-CLAM [RSG98] could be connected as well. Since both systems work with a planning architecture it would be possible to incorporate their partial plans into our architecture.

In the current implementation the planning cell manager has at its disposal the TPS-, LEO-, FO-, and CAS-agents, as well as ΩMEGA's proof planner and interactive proof development facilities.

6 Examples Revisited

We illustrate now how the examples introduced in 2.4 are tackled in our system architecture. We want to point out that our system is still under development and that the following examples can currently be solved semi-automatically. In particular, the examples have been analysed in a simulation of our system, where the user took over the crucial tasks of the planning cell manager.

6.1 Example I

$$\text{(I)} \qquad \begin{array}{c}[\forall f_{\blacksquare}\forall g_{\blacksquare}(f' = g' \wedge \exists x_{\blacksquare} f(x) = g(x)) \Leftrightarrow f = g] \\ \Rightarrow \forall n_{\blacksquare}(\lambda x_{\blacksquare} n \cdot \log(x) = \lambda x_{\blacksquare} \log(x^n))\end{array}$$

In the first run theorem (I) is simplified by ΩMEGA's proof planner and by TPS by applying the obvious natural deduction rules as far as possible. As a result, both systems pass the following sub-problems to the planning cell manager:

$$(\lambda x_{\blacksquare} n \cdot \log(x))' = (\lambda x_{\blacksquare} \log(x^n))' \tag{1}$$

and

$$\exists x_{\blacksquare} n \cdot \log(x) = \log(x^n) \tag{2}$$

Both sub-problems are then tackled independently in the subsequent concurrent runs. Sub-problem (1) can be solved by both, the computer algebra system MAPLE and ΩMEGA's proof planner, using the knowledge provided by the domain server. The domain server contains the appropriate theorems that determine the respective derivatives. Hence, the planning cell manager has the choice between two different solutions for the same problem[3].

The task in the sub-problem (2) is to compute a witness-term that satisfies the given equality. In our system this is currently only solvable by MAPLE. When asked for which x the equality $n \cdot \log(x) = \log(x^n)$ holds, MAPLE simply returns x itself. This indicates that the two functions are equal within their domain. Hence, the planning cell manager now has to chose a value from the domain of the logarithmic functions and call MAPLE to do the necessary simplifications. In our case $x = 1$ is the obvious choice, since for this value the left and right hand side of the equation evaluate to 0 (this requires knowledge like $\log(1) = 0$).

[3] In our case, the system would prefer the solution proposed by the planner since it is based on the knowledge of the domain server. The knowledge of the domain server is considered more reliable than the information from an external component.

6.2 Example II

(II) $$\forall n_{\blacksquare}\forall a_{\blacksquare}(a^n \circ a) = (a \circ a^n)$$

In a first run ΩMEGA's proof planner or the user employs an induction method in order to split the theorem into two subgoals: the trivial base case $\forall a_{\blacksquare}(a \circ a) = (a \circ a)$ and the more complicated step case $(\forall a_{\blacksquare}(a^n \circ a) = (a \circ a^n)) \Rightarrow (\forall a_{\blacksquare}((a^n \circ a) \circ a) = (a \circ (a^n \circ a)))$. In a subsequent run the higher-order extensionality reasoner LEO works on this example. First the definition of $\circ$ is unfolded and as LEO is a resolution based reasoner the problem is then negated and reduced to clause normal-form. Hence, LEO's initial clause set consists of the following two clauses:

$$\neg((\lambda x,y_{\blacksquare}\exists u_{\blacksquare}(\exists v_{\blacksquare}a^n(x,v) \wedge a(v,u)) \wedge a(u,y)) \;=\; (\lambda x,y_{\blacksquare}\exists u_{\blacksquare}a(x,u) \wedge (\exists v_{\blacksquare}a^n(u,v) \wedge a(v,y))))$$

and

$$(\lambda x,y_{\blacksquare}\exists z_{\blacksquare}a^n(x,z) \wedge v(z,y)) = (\lambda x,y_{\blacksquare}\exists z_{\blacksquare}a(x,z) \wedge v^n(z,y)))$$

where a is a Skolem constant and, in the second formula, v a free variable. By applying its special extensionality rules LEO generates the following set of first-order clauses from the negated unit equation (note that LEO always applies its extensionality rules exhaustively in a straightforward way to equations between set and relation descriptions as given here), where $s^1,\ldots,s^6$ are Skolem terms and x,y,z,w are variables:

$$a(s^1,s^2) \vee a(s^3,s^2) \tag{1}$$
$$a^n(s^4,s^1) \vee a(s^3,s^2) \tag{2}$$
$$a(s^5,s^4) \vee a(s^3,s^2) \tag{3}$$
$$a(s^1,s^2) \vee a(s^6,s^3) \tag{4}$$
$$a^n(s^4,s^1) \vee a(s^6,s^3) \tag{5}$$
$$a(s^5,s^4) \vee a(s^6,s^3) \tag{6}$$
$$a(s^1,s^2) \vee a^n(s^5,s^6) \tag{7}$$
$$a^n(s^4,s^1) \vee a^n(s^5,s^6) \tag{8}$$
$$a(s^5,s^4) \vee a^n(s^5,s^6) \tag{9}$$
$$a^n(s^4,s^1) \vee \neg a(x,s^2) \vee \neg a^n(y,x) \vee \neg a(s^5,y) \tag{10}$$
$$a(s^5,s^4) \vee \neg a(x,s^2) \vee \neg a^n(y,x) \vee \neg a(s^5,y) \tag{11}$$
$$a(s^3,s^2) \vee \neg a(x,s^2) \vee \neg a(y,x) \vee \neg a^n(s^5,y) \tag{12}$$
$$a(s^6,s^3) \vee \neg a(x,s^2) \vee \neg a(y,x) \vee \neg a^n(s^5,y) \tag{13}$$
$$a^n(s^5,s^6) \vee \neg a(x,s^2) \vee \neg a(y,x) \vee \neg a^n(s^5,y) \tag{14}$$
$$a(s^1,s^2) \vee \neg a(x,s^2) \vee \neg a^n(y,x) \vee \neg a(s^5,y) \tag{15}$$
$$\neg a(x,s^2) \vee \neg a(y,x) \vee \neg a^n(s^5,y) \vee \neg a(z,s^2) \vee \neg a^n(w,z) \vee \neg a(s^5,w) \tag{16}$$

Analogously, LEO transforms the positive unit equation into the following set of clauses, where s^7 and s^8 are Skolem terms:

$$v(s^7,x) \vee \neg v^n(y,x) \vee \neg v(z,y) \tag{17}$$
$$v^n(x,s^7) \vee \neg v^n(y,z) \vee \neg v(x,y) \tag{18}$$
$$v^n(s^8,x) \vee \neg v(y,x) \vee \neg v^n(z,y) \tag{19}$$
$$v(x,s^8) \vee \neg v(y,z) \vee \neg v^n(x,y) \tag{20}$$

Thus, within less than 3 seconds this inconsistent set of clauses is available within LEO. Although the inconsistency of this set of generated clauses can be proved trivially in a specialised

first-order reasoner (OTTER needs less than 2 seconds when employing the higher-order to first-order encoding as inbuilt to ΩMEGA), this task takes quite a long time within the higher-order reasoner LEO (more than 3 minutes). This is due to the fact that LEO is not specialised in first-order reasoning, and it also has to cope with other – potentially higher-order – clauses in its search space. Fortunately, our architecture enables LEO to dynamically exchange the clauses between the reasoners in the centre of expertise for first-order logic. This is done via the communication blackboard maintained by the plan server. Hence, the generated set of clauses is passed to these specialised reasoners which detect the contradiction immediately.

6.3 Example III

(III) $\forall p_{\bullet} \mathit{partition}(p) \Rightarrow (\exists q_{\bullet} \mathit{equivalence\text{-}rel}(q) \wedge (\mathit{equivalence\text{-}classes}(q) = p))$

The search for the solution of theorem (III) has not, as said in 2.4, been automated yet. However, we identified the tasks that need to be done. Here we give a general analysis of these.

First, the initial proof goal is split into three subgoals (e.g., with the help of a proof planner). Namely, from a given partition p we can derive the existence of an equivalence relation q, constituting the first subgoal. For the same equivalence relation q it holds that its equivalence classes are exactly p. The two directions of the set equality give us the second and the third subgoal. Next, higher-order equality and extensionality reasoning is required. The first two subgoals can be solved automatically by the higher-order prover TPS. The last subproblem, which requires a fair amount of extensionality reasoning, we expect to be solvable by cooperation between the higher-order extensionality prover LEO and a first-order automated theorem prover. LEO provides the necessary higher-order extensionality treatment, however, it cannot cope with the large number of first-order clauses that are generated subsequently. Therefore, this set of clauses is passed via ΩMEGA to the first-order specialist available within our concurrent subsystems society.

6.4 Evaluation

Although the examples presented here are non-trivial with respect to their automation, they are still far away from being impressive from a mathematical point of view. The main point here is not to demonstrate how powerful our system already is, but to illustrate that the architecture is capable of embedding different styles of automated reasoning and computation in a sensible way into one single system. Above all, it allows to integrate the user on two different interaction layers: firstly, the user may operate as an intelligent proof planning agent in an interactive instance of ΩMEGA in competition to the other reasoning agents in each running phase. Secondly, the user may supply expert knowledge and support to the planning cell manager by (re-)directing the system in promising directions. One of our aims is to relocate necessary user interaction in our system mainly to the latter layer. Note that the above examples can already be proved without any user interaction on the former layer.

Another interesting aspect illustrated by theorem (II) is that the system has capabilities to reduce problems that are essentially of first-order nature, but which are stated in our system in a more elegant and natural (and thus user-friendly) higher-order formulation, automatically into an appropriate first-order encoding. This contrasts reasoning in a restricted first-order environment where the encoding of a problem into a first-order formulation has to be supplied in advance by the user. Burdening the user with first-order coding is problematic for two reasons. First, the encoding requires a fair amount of explicit knowledge, and second, it often requires the user to solve some crucial aspects of the problem, for instance, by implicitly employing extensionality reasoning.

7 Related Work

Our work is related to work in multi-agent systems in general, and to approaches to distributed proof search and agent-oriented theorem proving in special.

One of the first theorem proving systems which uses distributed search is Denzinger's "team work" approach, in which theorem proving is modelled as a team of experts. A team has a supervisor which distributes the tasks, and a number of experts which work on selected subtasks and which are evaluated by referees. Important results are exchanged in team meetings, and based on the referee reports the team may be restructured (see, e.g., [DF94]). While theorem proving in the original team work system can be viewed as selecting strategies within multiple instances of the same theorem prover, this work has been extended in the TECHS system in order to realise the cooperation between a set of heterogeneous first-order theorem provers (see, e.g., [DF99]). Partial results are exchanged between the different theorem provers in form of clauses, and different referees filter the communication at the sender and receiver side of the communication.

There are two main differences between Denzinger *et al*'s and our work. First, the diversity of integrated systems is bigger in our case (incorporating higher-order theorem provers and computer algebra systems). Second, we try to integrate reactive and deliberative behaviour and have a more open-ended approach to search with our focused search approach, at the price that our model has a more complex structure, but may turn out to be more flexible. In particular, TECHS does not provide techniques to translate selected results of the reasoning agents (e.g., clauses derived by a first-order theorem prover) into derivations in a uniform proof data structure, whereas for most systems in our approach this is possible. Hence, our evaluation criteria may be able to exploit knowledge on a more abstract level and relate the contributions of the agents to the current partial proof in the global proof attempt. As our system stores interesting results of previous running phases it is able to backtrack (i.e., to relocate the focus) to previously dismissed search directions. This kind of focus shifts are not supported in the TECHS approach.

The concurrent resource managed approach to theorem proving is related to agent based theorem proving architectures. Some of the state of the art agent architectures for automatic and interactive theorem proving are discussed in [Fis97], [BmS98], and [BmS99].

To give a full account of related work in multi-agent systems would go much beyond the scope of this paper. We borrow in our approach much from the general terminology used in multi-agent systems – expressions like plan, reactive behaviour, resources are very common in this field. Important in our context is the development from planning, as described for instance in STRIPS [FN71], over an approach of merely reactive behaviour, as advocated by [Bro91], to very sophisticated multi-level architectures, as presented by [Slo99].

From an architectural point of view our system can be seen as an adaptation of the *configuration for multiple planning cells* of the multi-agent planning architecture (as described by [WM98]) to the proof planning domain. MPA is a distributed architecture which can run a number of agents in parallel. It is based on the concept of planning cells. Each planning cell consists of a collection of agents, two of which are distinguished: the planning cell manager and the plan server. Agents employed in MPA have the capability to communicate directly with other agents. This is facilitated by wrappers, agent libraries, and communication and plan transformations which define the message protocols. MPA can be configured in two ways: as a single planning cell configuration for generating individual solutions to a planning task, or as a multiple planning cell configuration for generating alternative solutions in parallel. A multiple planning cell configuration includes multiple instances of the single planning cell configuration coordinated by the meta planning cell manager. The MPA framework has been used in the development of several large-scale problem solving systems.

There are clearly many similarities between our concurrent resource managed deduction architecture and MPA. Like MPA, our system is composed of planning cells which consist of a planning cell manager, a plan server and a number of specialised reasoning agents. When triggered, these agents run in parallel in both systems. We can view our architecture as an adapted example of a multiple planning cell configuration. However, unlike in MPA which is applied to the domain of planning, the problem domain tackled by our architecture is proof planning and deduction. Clearly, this means that our specialised problem solvers are different from the ones employed in MPA. Other differences between the two frameworks are that in MPA agents can communicate directly with each other, whereas in our system the communication is indirect via a communication blackboard provided by the plan server. Furthermore, MPA employs a shared plan representation

which is understood by all agents, whereas our agents are used as glass boxes which use their own representation. We employ the ΩMEGA proof data structure as a uniform data structure used by the planning cell shell, and the planning manager to carry out the translation to and from the data structures of individual agents.

The comparison of our work with MPA [WM98] also clarifies the commonalities and differences of our approach to the approach of Melis and Meier, which is another related system. Namely, the system described in [MM99] can be viewed as an adaptation of the *single planning cell configuration* in MPA, applied to the proof planning domain, rather than *multiple planning cell configuration* as it is in our case.

Recent work in multi-agent systems tries to reconcile different paradigms by integrating them in a hybrid system, in particular to combine plan-oriented and reactive methods. We do not claim that we have achieved such a combination with our system yet. In order to achieve this, the critical question of how to allocate resources automatically would need to be resolved, and a deeper understanding of the interplay of reactivity and long-term planning would be needed. We think, however, that we made first steps to transfer and implement the ideas in this general area. In particular, we combine heterogeneous approaches like theorem proving in first-order logic, in higher-order logic, computer algebra, proof planning, and interactive theorem proving. The integration of further components (like model generators) should be straightforward. In order to further develop the full potential of the approach, many problems, some of which are very hard, have to be solved. We briefly discuss some in the next section.

8 Further Work

The most crucial open research task concerns further development of the planning cell manager which controls and realises the focused proof search of the overall system.

Clearly, the success of the overall system strongly depends on the availability and applicability of suitable evaluation and assessment criteria. Although some sensible criteria have been presented in the paper (simplicity/complexity of partial proofs, the theory/logic a subproblem belongs to, and the similarity of open subproblems to already solved problems stored in the database), they need to be extended by more fine-grained ones.

Despite our aim to increase the degree of automation and the quality of the planning cell manager in the evaluation and assessment processes, we explicitly want to integrate the user as final means of control for these crucial tasks. Ideally, the planning cell manager presents the results of its evaluation and assessment processes in an appropriate user interface. The user will then be able to bring into the system his/her expertise in order to (re-)direct the focus of search to the most promising directions. In order to support this way of user interaction, the corresponding graphical user interface facilities have to be built. Therefore, we want to appropriately extend ΩMEGA's graphical user interface LOUI. For example, LOUI should be able to present several partial proof plans – the most promising ones returned in each running phase of the system.

Currently, we use ΩMEGA's database of mathematical theories as a domain server. As soon as the development of MBASE is at an acceptable stage, MBASE will be employed as domain server in our system instead.

The MATHWEB environment currently being developed will provide a fruitful basis for our system. However, only some of the reasoning agents exist as MATHWEB-agents. Furthermore, the required communication and resource management facilities of the MATHWEB-agent are still not fully available. Therefore, we employ in our prototype system ad hoc implementations of agent shells. Subsequently, these ad hoc agent shells will be replaced by the respective MATHWEB-shells.

Interesting further work will also concentrate on the question of whether it is possible and sensible to employ copies of the system architecture at a lower layer, so that the system allows, for instance, grouping of homogeneous provers tackling similar kinds of problems into one single reasoning agent. It may be useful to group classical first-order reasoners together to form a centre of expertise for classical first-order logic. Ideally, such centres of expertise may use a mechanism analogous to the overall system in order to organise the communication between its subsystems,

and to further distribute the resources they obtain at the upper level. The systems in a centre of expertise can be evaluated using fine-grained evaluation criteria. Evaluation experiments of this kind have been carried out in the past, for instance, for first-order theorem provers and other homogeneous systems, cf. [Fis97, Wol98, DF99]. They proved to be successful and gave positive results. That is, it is possible to realise a more homogeneous system communication within the centre of expertise. Furthermore, the centres of expertise can have a dynamic nature, that is, they might remodel themselves differently for different problem domains or explicitly learn in which areas their particular strengths and weaknesses are.

Our current working example is theorem (III). The need for cooperation of specialised reasoning systems, and some concrete cooperation ideas to solve this challenging problem are illustrated in [BBS99]. Our system architecture now provides the required basis for tackling this example, even though it has not yet been shown that it can be solved automatically in our approach. The bottleneck is the LEO system, whose still rather prototypical implementation is not able to plunge deep enough into the search space when tackling the crucial subproblem requiring a non-trivial amount of extensionality reasoning. More precisely, LEO is not able to apply its special extensionality reasoning facilities in such a way that the subsequently generated set of first-order clauses (to be passed to a centre for first-order reasoning) is contradictory. Thus, one of our future tasks will be to increase the reasoning power of some of the reasoning agents exploited in our system.

9 Conclusion

We described in this paper an approach to reasoning with a concurrent resource management based architecture incorporated into a proof planning framework. As discussed here, such an architecture gives scope to improved mechanised reasoning capabilities. Unlike a conventional distributed parallel model of theorem proving, a concurrent resource managed architecture provides a paradigm where the communication between the underlying systems and the management of resources for these reasoners can be realised. The resource management technique results in a less brittle search, and hence promises to find more results than traditional search techniques. We have demonstrated the feasibility and usefulness of our approach by proving a few theorems with our prototype implementation. The hope is that such a system will be able to prove a significant number and range of theorems that have previously not been proved automatically and to ease the task of interactive theorem proving in many cases as well.

References

[ABI+96] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[AHMS99] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. System Description: inka 5.0 - A Logic Voyager. In H. Ganzinger, editor, *Proceedings of the 16th Conference on Automated Deduction (CADE–16)*, volume 1632 of *LNAI*, pages 207–211, Trento, Italy, 1999. Springer Verlag, Berlin, Germany.

[BA98] Matthew Bishop and Peter B. Andrews. Selectively instantiating definitions. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th Conference on Automated Deduction (CADE–15)*, volume 1421 of *LNAI*, pages 365–380, Lindau, Germany, July 1998. Springer Verlag, Berlin, Germany.

[BBS99] Christoph Benzmüller, Matthew Bishop, and Volker Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5(3):188–207, March 1999. Special issue on Integration of Deduction System.

[Ben99] Christoph Benzmüller. *Equality and Extensionality in Automated Higher-Order Theorem Proving.* PhD thesis, Faculty of Technology, Saarland University, 1999.

[Ble86] W. W. Bledsoe. Some thoughts on proof discovery. In Robert Keller, editor, *Proceedings of the IEEE Symposium on Logic Programming*, pages 2–10, Salt Lake City, Utah, USA, September 1986. IEEE Computer Society.

[BmCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction (CADE–14)*, volume 1249 of *LNAI*, pages 252–255, Townsville, Australia, 1997. Springer Verlag, Berlin, Germany.

[BmK98] Christoph Benzmüller and Michael Kohlhase. LEO – a higher order theorem prover. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th Conference on Automated Deduction (CADE–15)*, volume 1421 of *LNAI*, pages 139–144, Lindau, Germany, July 1998. Springer Verlag, Berlin, Germany.

[BmS98] Christoph Benzmüller and Volker Sorge. A Blackboard Architecture for Guiding Interactive Proofs. In F. Giunchiglia, editor, *Artificial Intelligence: Methodology, Systems and Applications, Proceedings of the of the 8th International Conference AIMSA'98*, number 1480 in LNAI, pages 102–114, Sozopol, Bulgaria, October 1998. Springer Verlag, Berlin, Germany.

[BmS99] Christoph Benzmüller and Volker Sorge. Critical Agents Supporting Interactive Theorem Proving. In P. Barahona and J. J. Alferes, editors, *Progress in Artificial Intelligence, Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA-99)*, volume 1695 of *LNAI*, pages 208–221, Évora, Portugal, 21–24, September 1999. Springer Verlag, Berlin, Germany.

[Bro91] Rodney A. Brooks. Intelligence without reason. Memo No. 1293, MIT, April 1991.

[Bun88] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE–9)*, volume 310 of *LNCS*, Argonne, IL, USA, 1988. Springer Verlag, Berlin, Germany.

[Bun94] Alan Bundy. A subsumption architecture for theorem proving? *Philosophical Transactions of the Royal Society of London*, **349/1689**:71–85, 1994.

[BvHHS90] Alan Bundy, Frank van Harmelen, Christian Horn, and Alan Smaill. The OYSTER-CLAM system. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction (CADE–10)*, volume 449 of *LNCS*, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany.

[CS99] Lassaad Cheikhrouhou and Volker Sorge. $\mathcal{PDS}$ — A Three-Dimensional Data Structure for Proof Plans. Submitted to the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000), 1999.

[Dah97] Ingo Dahn. Integration of Automated and Interactive Theorem Proving in ILF. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction (CADE–14)*, volume 1249 of *LNAI*, pages 57–60, Townsville, Australia, July 13–17 1997. Springer Verlag, Berlin, Germany.

[DF94] Jörg Denzinger and Matthias Fuchs. Goal oriented equational theorem proving using team work. In Bernhard Nebel and Leonie Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence – Proceedings of KI-94, 18th German Annual Conference on Artificial Intelligence*, pages 343–354, Saarbrücken, Germany, 1994. Springer Verlag, Berlin, Germany, LNAI 861.

[DF99] Jörg Denzinger and Dirk Fuchs. Cooperation of Heterogeneous Provers. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 10–15, Stockholm, Sweden, 1999.

[DKS97] Jörg Denzinger, J. Kronenburg, and Stephan Schulz. DISCOUNT - A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[FF94] T. Finin and R. Fritzson. KQML - A language and protocal for knowledge and information exchange. In *Proceedings of the 13th International Distributed AI Workshop*, pages 127–136, Seattle, 1994.

[FHJ+99] A. Franke, S. Hess, Ch. Jung, M. Kohlhase, and V. Sorge. Agent-Oriented Integration of Distributed Mathematical Services. *Journal of Universal Computer Science*, 5(3):156–187, March 1999. Special issue on Integration of Deduction System.

[Fis97] Michael Fisher. An Open Approach to Concurrent Theorem Proving. In J. Geller, H. Kitano, and C. Suttner, editors, *Parallel Processing for Artificial Intelligence*, volume 3. Elsevier/North Holland, 1997.

[FK99] Andreas Franke and Michael Kohlhase. MBase: Representing mathematical Knowledge in a Relational Data Base. In Alessandro Armando and Tudor Jebelean, editors, *CALCULEMUS 99, Systems for Integrated Computation and Deduction*, Electronic Notes in Theoretical Computer Science, pages 135–152, Trento, Italy, July 11–12 1999. Elsevier. URL: `http://www.elsevier.nl/locate/entcs`.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**:189–208, 1971.

[Had44] Jacques Hadamard. *The Psychology of Invention in the Mathematical Field*. Dover Publications, New York, USA; edition 1949, 1944.

[HF97] Xiaorong Huang and Armin Fiedler. Proof Verbalization in *PROVERB*. In J. Siekmann, F. Pfenning, and X. Huang, editors, *Proceedings of the First International Workshop on Proof Transformation and Presentation*, pages 35–36, Schloss Dagstuhl, Germany, 1997.

[Hua94] Xiaorong Huang. Reconstructing Proofs at the Assertion Level. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *LNAI*, pages 738–752, Nancy, France, 1994. Springer Verlag, Berlin, Germany.

[KKS98] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[McC97] W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, December 1997.

[Mei97] Andreas Meier. Übersetzung automatisch erzeugter Beweise auf Faktenebene. Master's thesis, Computer Science Department, Universität des Saarlandes, Saarbrücken, Germany, 1997.

[Mel95] Erica Melis. A Model of Analogy-Driven Proof-Plan Construction. In Chris S. Mellish, editor, *Proceedings of the 14th International Joint Conference on A rtificial Intelligence (IJCAI)*, pages 182–189, Montreal, Canada, 1995. Morgan Kaufmann, San Mateo, California, USA.

[Min85] Marvin Minsky. *The Society of Mind*. Simon & Schuster, New York, USA, 1985.

[MM99] Erica Melis and Andreas Meier. Proof planning with multiple strategies ii. In Bernhard Gramlich, Helene Kirchner, and Frank Pfenning, editors, *Proceedings of Workshop on Stategies in Automated Deduction (STRATEGIES'99)*, 1999.

[Pea84] Judea Pearl. *Heuristics – Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, Massachusetts, USA, 1984.

[Pól65] George Pólya. *Mathematical Discovery – On Understanding, Learning, and Teaching Problem Solving.* Princeton University Press, Princeton, New Jersey, USA, 1962/1965. Two volumes, also as combined edition, 1981, John Wiley and Sons, New York, USA.

[Red98] Darren Redfern. *The Maple Handbook: Maple V Release 5.* Springer Verlag, Berlin, Germany, 1998.

[RN95] Stuart J. Russell and Peter Norvig. *Artificial Intelligence – A Modern Approach.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.

[RSG98] Julian D.C. Richardson, Alan Smaill, and Ian M. Green. System description: Proof planning in higher-order logic with λCLAM. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th Conference on Automated Deduction (CADE–15)*, volume 1421 of *LNAI*, pages 129–133, Lindau, Germany, July 1998. Springer Verlag, Berlin, Germany.

[SHB+99] Jörg Siekmann, Stephan M. Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet, and Volker Sorge. $\mathcal{LOUI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11(3):326–342, 1999.

[Sie96] J. Siegel. *Corba: Fundamentals and Programming.* John Wiley & Sons Inc., 1996.

[Slo99] Aaron Sloman. Architectural requirements for human-like agents – both natural and artificial. (what sorts of machines can love?). In Kerstin Dautenhahn, editor, *Human Cognition and Social Agent Technology.* John Benjamins Publishing, 1999.

[Ull00] Carsten Ullrich. Analogie als Strategie im Beweisplanen. Master's thesis, Computer Science Department, Universität des Saarlandes, Saarbrücken, Germany, 2000. Forthcoming.

[Wae64] Bartel L. van der Waerden. Wie der Beweis der Vermutung von Baudet gefunden wurde. *Abh. Math. Sem. Univ. Hamburg*, **28**:6–15, 1964.

[WM98] David E. Wilkins and Karen L. Myers. A Multiagent Planning Architecture. In Reid Simmons, Manuela Veloso, and Stephen Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 154–162, Pittsburgh, PEN, USA, June 7–10 1998. AAAI Press, Menlo Park, CA, USA.

[Wol98] Andreas Wolf. P-SETHEO: Strategy Parallelism in Automated Theorem Proving. In Harrie de Swart, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-98)*, volume 1397 of *LNAI*, page 320, Oisterwijk, The Netherlands, May 5–8 1998. Springer-Verlag.

[Zil95] S. Zilberstein. Models of Bounded Rationality. In *AAAI Fall Symposium on Rational Agency*, Cambridge, Massachusetts, November 1995.