Critical Agents Supporting Interactive Theorem Proving

Christoph Benzmüller and Volker Sorge

Fachbereich Informatik, Universität des Saarlandes,
D-66123 Saarbrücken, Germany
{chris|sorge}@ags.uni-sb.de
http://www.ags.uni-sb.de/ chris|sorge

Abstract. We introduce a resource adaptive agent mechanism which supports the user of an interactive theorem proving system. The mechanism, an extension of [4], uses a two layered architecture of agent societies to suggest applicable commands together with appropriate command argument instantiations. Experiments with this approach show that its effectiveness can be further improved by introducing a resource concept. In this paper we provide an abstract view on the overall mechanism, motivate the necessity of an appropriate resource concept and discuss its realization within the agent architecture.

1 Introduction

Interactive theorem provers have been developed to overcome the shortcomings of purely automatic systems and are typically applied in demanding domains where fully automated techniques usually fail. Interaction is needed, for example, to speculate lemmata or to guide and control the reasoning, for instance, by providing the crucial steps in a complicated proof attempt.

Typical tactic-based interactive theorem proving systems such as HOL [14], TPS [2], or our own ΩMEGA [3] offer expressive problem formulation and communication languages and employ human oriented calculi (e.g., a higher-order natural deduction or sequent calculus) in order to keep both proof and proof construction comprehensible.

Initially, problems are given as a theorem together with a set of axioms. Proofs are then constructed by successive application of tactics which are either rules from the given calculus or little procedures that apply sequences of such rules (cf. [13]). Generally, the user can employ a tactic by invoking an associated command. Tactics can be applied forward to axioms and derived facts or backward to open problems which may result in one or several new open problems. A proof is complete when no open subgoal remains, i.e. when the originally given theorem is successfully justified by a derivation from the given axioms. In most systems the user can easily combine existing tactics in order to build new, possibly more abstract ones. Moreover, some systems offer the use of external reasoning components such as automated theorem provers or computer algebra systems in order to enhance their reasoning power.

The number of tactics (and therefore the number of commands) offered to the user by an interactive theorem prover is often quite large. Thus, it is important to support the user (especially the non-expert user) in selecting the *right* command together with appropriate instantiations for its parameters (e.g., proof lines, terms, or sub-term positions) in each proof step.

Although suggestion mechanisms are already provided in state of the art interactive theorem provers, they are still rather limited in their functionality as they usually

(i) use inflexible sequential computation strategies,
(ii) do not have anytime character,
(iii) do not work steadily and autonomously in the background of a system, and
(iv) do not exhaustively use available computation resources.

In order to overcome these limitations we proposed in [4] a new, flexible support mechanism with anytime character. It suggests commands, applicable in the current proof state — more precisely commands that invoke applicable tactics — together with suitable argument instantiations[1]. It is based on two layers of societies of autonomous, concurrent agents which steadily work in the background of the system and dynamically update their computational behavior to the state of the proof and/or specific user queries to the suggestion mechanism. By exchanging relevant results via blackboards the agents cooperatively accumulate useful command suggestions which can then be heuristically sorted and presented to the user.

A first implementation of the support mechanism in the ΩMEGA-system yielded promising results. However, experience showed that the number of agents can become quite large and that some agents perform very costly computations, such that the initial gain of the distributed architecture and the use of concurrency is easily outweighed by the mechanism's computational costs. In a first step to overcome this dilemma we developed a resource adapted concept for the agents in order to allow for efficient suggestions even in large examples. However, the concurrent nature of the mechanism provides a good basis to switch from a static to a dynamic, resource adaptive control of the mechanism's computational behavior[2]. Thereby, we can exploit both knowledge on the prior performance of the mechanism as well as knowledge on classifying the current proof state and single agents in order to distribute resources.

After giving an example in the next section, to which we will refer throughout this paper, we review in Sec. 3 our two layered agent mechanism as introduced

[1] Whereas in [4] and in this paper the suggestion mechanism is described with respect to tactical theorem proving based on a ND-calculus [11], we want to point out that our mechanism is in no way restricted to a specific logic or calculus, and can easily be adapted to other interactive theorem proving contexts as well.

[2] In this paper we adopt the notions of *resource adapted* and *resource adaptive* as defined in [20], where the former notion means that agents behave with respect to some initially set resource distribution. According to the latter concept agents have an explicit notion of resources themselves, enabling them to actively participate in the dynamic allocation of resources.

in [4]. In Sec. 4 we present a static resource concept to enhance the mechanism. This concept is then extended in Sec. 5 into a resource adaptive one, where the resource allocations are dynamic and based on the following criteria:

1. The lower layer agents monitor their own contributions and performance in the past in order to estimate the fruitfulness of their future computations.
2. The resource allocations of the societies of lower layer agents is dynamically monitored and adjusted on the upper layer.
3. A *classification agent* gathers explicit knowledge about the current proof state (e.g., which theory or which logic the current subgoal belongs to) and passes this information to the lower layer agents.

Hence, the agents in our mechanism have a means to decide whether or not they should pursue their own intentions in a given proof state. Their decision is based on sub-symbolic (1 and 2) as well as on on symbolic information (3). We finally conclude by discussing what a state of the art interactive theorem prover can gain from employing the proposed suggestion mechanism and by hinting at possible future work.

2 Reference Example

In the remainder of this paper we will frequently refer to the proof of the higher order (HO) theorem $(p_{o\to o}\ (a_o \wedge b_o)) \Rightarrow (p\ (b \wedge a))$, where o denotes the type of truth values. Informally this example states: If the truth value of $a \wedge b$ is element of the set p of truth values, then the value of $b \wedge a$ is also in p. Alternatively one can read the problem as follows: Whenever a unary logical operater p maps the value of $a_o \wedge b_o$ to *true*, then this also holds for the value of $b \wedge a$. Although, the theorem looks quite simple at a first glance this little higher-order (HO) problem cannot be solved by most automatic HO theorem provers known to the authors, since it requires the application of the extensionality principles which are generally not built-in in HO theorem proving systems. However, within the ΩMEGA-system [3] this problem can easily be proven partially interactively and automatically.

ΩMEGA employs a variant of Gentzen's natural deduction calculus (ND) [11] enriched by more powerful proof tactics and the possibility to delegate reasonably simple sub-problems to automated theorem provers. Thus, the following proof for the example theorem can be constructed[3]:

L_1	$(L_1) \vdash (p\ (a \wedge b))$	Hyp
L_4	$(L_1) \vdash (b \wedge a) \Leftrightarrow (a \wedge b)$	OTTER
L_3	$(L_1) \vdash (b \wedge a) = (a \wedge b)$	$\Leftrightarrow 2{=}: (L_4)$
L_2	$(L_1) \vdash (p\ (b \wedge a))$	$=_{\text{subst}}: (\langle 1 \rangle)(L_1 L_3)$
C	$() \vdash (p\ (a \wedge b)) \Rightarrow (p\ (b \wedge a))$	$\Rightarrow_I: (L_2)$

[3] Linearized ND proofs are presented as described in [1]. Each proof line consists of a label, a set of hypotheses, the formula and a justification.

The idea of the proof is to show that the truth value of $a \wedge b$ equals that of $b \wedge a$ (lines L_3 and L_4) and then to employ equality substitution (line L_2). The equation $(b \wedge a) = (a \wedge b)$ is derived by application of boolean extensionality from the equivalence $(b \wedge a) \Leftrightarrow (a \wedge b)$ in line L_3, whereas the rewriting step in line L_2 is indicated in the line's justification which reads as 'substitute the subterm at position $\langle 1 \rangle$ in line L_1 according to the equation stated in line L_3' where position $\langle 1 \rangle$ corresponds to the first argument of the predicate p. The equivalence $(b \wedge a) \Leftrightarrow (a \wedge b)$ in line L_3, can either be proven interactively or, as in the given proof, justified by the application of the first-order prover OTTER [16] which hides the more detailed subproof.

Our agent mechanism is able to suggest all the single proof steps together with the respective parameter instantiations to the user. In particular, for the proof of line L_4 it suggests the choice between the application of OTTER or the next interactive prove step to the user. In case of the latter choice, the mechanism's further suggestions can be used to finish the proof completely interactively.

In the remainder of this paper we use the proof of the presented example in this section to demonstrate the working scheme of the suggestion mechanism and to motivate the incorporation of resource concepts.

3 Suggesting Commands

The general suggestion mechanism is based on a two layered agent architecture displayed in Fig. 1 which shows the actual situation after the first proof step in the example, where the backward application of $\Rightarrow_I$ introduces the line L_1 as new hypothesis and the line L_2 as the new open goal. The task of the bottom layer of agents (cf. the lower part of Fig. 1) is to compute possible argument instantiations for the provers commands in dependence of the dynamically changing partial proof tree. The task of the top layer (cf. the upper part of Fig. 1) is to collect the most appropriate suggestions from the bottom layer, to heuristically sort them and to present them to the user.

The bottom layer consists of societies of argument agents where each society belongs to exactly one command associated with a proof tactic (a more formal notion of proof tactic is introduced in Sec. 3.1). On the one hand each argument agent has its own intention, namely to search in the partial proof for a proof line that suits a particular specification. On the other hand argument agents belonging to the same society also pursue a common goal, e.g., to cooperatively compute most complete argument suggestions (cf. the concept of partial argument instantiations in Sec. 3.1) for their associated command. Therefore the single agents of a society exchange their particular results via a suggestion blackboard and try to complete each others suggestions.

The top layer consists of a single society of command agents which steadily monitor the particular suggestion blackboards on the bottom layer. For each suggestion blackboard there exists one command agent whose intention is to

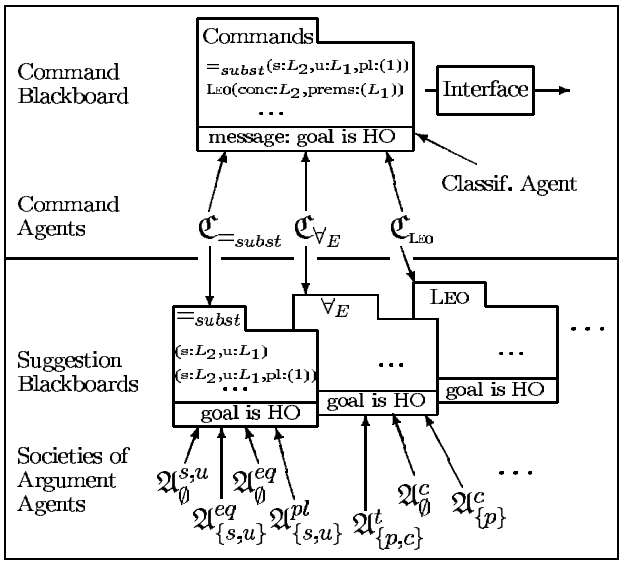

Fig. 1. The two layered suggestion mechanism.

determine the most complete suggestions and to put them on the command blackboard.

The whole distributed agent mechanism runs always in the background of the interactive theorem proving environment thereby constantly producing command suggestions that are dynamically adjusted to the current proof state. At any time the suggestions on the command blackboard are monitored by an interface component which presents them heuristically sorted to the user via a graphical user interface. As soon as the user executes a command the partial proof is updated and simultaneously the suggestion and command blackboards are reinitialized.

3.1 Partial Argument Instantiations

The data that is exchanged within the blackboard architecture heavily depends on a concept called a *partial argument instantiation* of a command. In order to clarify our mechanism we need to introduce this concept in detail.

In an interactive theorem prover such as ΩMEGA one has generally one command associated with each proof tactic that invokes the application of this tactic to a set of proof lines. In ΩMEGA these tactics have a fixed *outline*, i.e. a set of premise lines, conclusion lines and additional parameters, such as terms or term-positions. Thus the general instance of a tactic $\mathcal{T}$ can be formalized in the following way:

$$\frac{P_1 \cdots P_l}{C_1 \cdots C_m}\ \mathcal{T}(Q_1 \cdots Q_n),$$

where we call the P_i, C_j, Q_k the *formal arguments* of the tactic $\mathcal{T}$ (we give an example below).

We can now denote the command t invoking tactic $\mathcal{T}$ formally in a similar fashion as

$$\frac{p_{i_1} \cdots p_{i_{l'}}}{c_{j_1} \cdots c_{j_{m'}}} \; t(q_{k_1} \cdots q_{k_{n'}}),$$

where the formal arguments p_i, c_j, q_k of t correspond to a subset of the formal arguments of the tactic. To successfully execute the command some, not necessarily all, formal arguments have to be instantiated with *actual arguments*, e.g., proof lines. A set of pairs relating each formal argument of the command to an (possibly empty) actual argument is called a *partial argument instantiation (PAI)*.

We illustrate the idea of a PAI using the tactic for *equality substitution* $=_{Subst}$ and its corresponding command $=Subst$ as an example.

$$\frac{\Phi[x] \quad x = y}{\Phi'[y]} =_{Subst}(P^*) \qquad \longrightarrow \qquad \frac{u \quad eq}{s} =Subst(pl)$$

Here $\Phi[x]$ is an arbitrary higher order formula with at least one occurrence of the term x, P^* is a list of term-positions representing one or several occurrences of x in Φ, and $\Phi'[y]$ represents the term resulting from replacing x by y at all positions P^* in Φ. u, eq, s and pl are the corresponding formal arguments of the command associated with the respective formal arguments of the tactic. We observe the application of this tactic to line L_2 of our example:

$$\begin{array}{llll} L_1 & {}_{(L_1)}\vdash & (p\ (a \wedge b)) & \text{Hyp} \\ & \cdots & & \\ L_2 & {}_{(L_1)}\vdash & (p\ (b \wedge a)) & \text{Open} \end{array}$$

One possible PAI for $=Subst$ is the set of pairs $(u{:}L_1, eq{:}\epsilon, s{:}L_2, pl{:}\epsilon)$, where ϵ denotes the empty or unspecified actual argument. We omit writing pairs containing ϵ and, for instance, write the second possible PAI of the above example as $(u{:}L_1, s{:}L_2, pl{:}(\langle 1 \rangle))$. To execute $=Subst$ with the former PAI the user would have to at least provide the position list, whereas using the latter PAI results in the line L_3 of the example containing the equation.

3.2 Argument Agents

The idea underlying our mechanism to suggest commands is to compute PAIs as complete as possible for each command, thereby gaining knowledge on which tactics can be applied combined with which argument instantiations in a given proof state.

The main work is done by the societies of cooperating *Argument Agents* at the bottom layer (cf. Fig. 1). Their job is to retrieve information from the current proof state either by searching for proof lines which comply with the agents specification or by computing some additional parameter (e.g., a list of sub-term positions) with already given information. Sticking to our example we

can informally specify the agents $\mathfrak{A}^{u,s}_{\emptyset}$, $\mathfrak{A}^{eq}_{\emptyset}$, $\mathfrak{A}^{eq}_{\{u,s\}}$, and $\mathfrak{A}^{pl}_{\{u,s\}}$ for the $=\!Subst$ command (cf. [4] for a formal specification):

$$\begin{aligned}
\mathfrak{A}^{u,s}_{\emptyset} &= \left\{\begin{array}{l}\text{find an open line } u \text{ and a support line } s \text{ that differ}\\ \text{only wrt. occurrences of a single proper sub-term}\end{array}\right\}\\
\mathfrak{A}^{eq}_{\emptyset} &= \{\text{find a support line } eq \text{ which is an equation}\}\\
\mathfrak{A}^{eq}_{\{u,s\}} &= \left\{\begin{array}{l}\text{find a support line } eq \text{ which is an equation}\\ \text{suitable for rewriting } u \text{ into } s\end{array}\right\}\\
\mathfrak{A}^{pl}_{\{u,s\}} &= \{\text{compute the positions where } s \text{ and } u \text{ differ}\}
\end{aligned}$$

The attached superscripts specify the formal arguments of the command for which actual arguments are computed, whereas the indices denote sets of formal arguments that necessarily have to be already present in some PAI, so that the agent can carry out its own computations. For example agent $\mathfrak{A}^{eq}_{\{u,s\}}$ only starts working when it detects a PAI on the blackboard where actual arguments for u and s have been instantiated. On the contrary $\mathfrak{A}^{eq}_{\emptyset}$ does not need any additional knowledge in order to pursue its task to retrieve an open line containing an equation as formula.

The agents themselves are realized as autonomous processes that concurrently compute their suggestions and are triggered by the PAIs on the blackboard, i.e. the results of other agents of their society. For instance both agents, $\mathfrak{A}^{eq}_{\{u,s\}}$ and $\mathfrak{A}^{pl}_{\{u,s\}}$, would simultaneously start their search as soon as $\mathfrak{A}^{s,u}_{\emptyset}$ has returned a result. The agents of one society cooperate in the sense that they activate each other (by writing new PAIs to the blackboard) and furthermore complete each others suggestions.

Conflicts between agents do not arise, as agents that add actual parameters to some PAI always write a new copy of the particular PAI on the blackboard, thereby keeping the original less complete PAI intact. The agents themselves watch their suggestion blackboard (both PAI entries and additional messages) and running agents terminate as soon as the associated suggestion blackboard is reinitialized, e.g., when a command has been executed by the user.

The left hand side of Fig. 1 illustrates our above example: The topmost suggestion blackboard contains the two PAIs: $(u{:}L_1, s{:}L_2)$ computed by agent $\mathfrak{A}^{u,s}_{\emptyset}$ and $(u{:}L_1, s{:}L_2, pl{:}(\langle 1\rangle))$ completed by agent $\mathfrak{A}^{eq}_{\{u,s\}}$.

In the current implementation argument agents are declaratively specified. This strongly eases modification and enhancement of already given argument agents as well as the addition of new ones, even at run time.

3.3 Command Agents

In the society of *command agents* every agent is linked to a command and its task is to initialize and monitor the associated suggestion blackboards. Its intention is to select among the entries of the associated blackboard the most complete and appropriate PAI and to pass it, enriched with the corresponding command name to the command blackboard. That is, as soon as a PAI is written to the related

blackboard that has at least one actual argument instantiated, the command agent suggests the command as applicable in the current proof state, providing also the PAI as possible argument instantiations. It then updates this suggestion, whenever a *better* PAI has been computed. In this context *better* generally means a PAI containing more actual arguments. In the case of our example the current PAI suggested by command agent $\mathfrak{C}_{=Subst}$ is $(u{:}L_1, s{:}L_2, pl{:}(\langle 1 \rangle))$.

These suggestions are accumulated on a *command blackboard*, that simply stores all suggested commands together with the proposed PAI, continuously handles updates of the latter, sorts and resorts the single suggestions and provides a means to propose them to the user. In the case of the ΩMEGA-system this is achieved in a special command suggestion window within the graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [18]. The sorting of the suggestions is done according to several heuristic criteria, one of which is that commands with fully instantiated PAIs are always preferred as their application may conclude a whole subproof.

3.4 Experiences

Unfortunately, computations of single agents themselves can be very costly. Reconsider the agents of command $=Subst$: In ΩMEGA we have currently 25 different argument agents defined for $=Subst$ where some are computationally highly expensive. For example, while the agent $\mathfrak{A}_{\emptyset}^{eq}$ only tests head symbols of formulas during its search for lines containing an equation and is therefore relatively inexpensive, the agent $\mathfrak{A}_{\emptyset}^{u,s}$ performs computationally expensive matching operations. In large proofs agents of the latter type might not only take a long time before returning any useful result, but also will absorb a fair amount of system resources, thereby slowing down the computations of other argument agents.

[4] already tackles this problem partially by introducing a *focusing technique* that explicitly partitions a partial proof into subproblems in order to guide the search of the agents. This focusing technique takes two important aspects into account:

(i) A partial proof often contains several open subgoals and humans usually focus on one such subgoal before switching to the next.
(ii) Hypotheses and derived lines belonging to an open subgoal are chronologically sorted where the interest focuses on the more recently introduced lines.

Hence, the agents restrict their search to the actual subgoal (*actual focus*) and guide their search according to the chronological order of the proof lines.

4 Resource Adapted Approach

Since agents are implemented as independent threads a user can interrupt the suggestion process by choosing a command at any time without waiting for all possible suggestions to be made. An agent then either quits its computations regularly or as soon as it detects that the blackboard it works for has been

reinitialized, when the user has executed a command. It then performs all further computations with respect to the reinitialized blackboard. However, with increasing size of proofs some agents never have the chance to write meaningful suggestions to a blackboard. Therefore, these agents should be excluded from the suggestion process altogether, especially if their computations are very costly and deprives other agents of resources.

For this purpose we developed a concept of *static complexity ratings* where a rating is attached to each argument and each command agent, that roughly reflects the computational complexity involved for its suggestions. A *global complexity value* can then be adjusted by the user permitting to suppress computations of agents, whose ratings are larger than the specified value. Furthermore, commands can be completely excluded from the suggestion process. For example, the agent $\mathfrak{A}_{\emptyset}^{u,s}$ has a higher complexity rating than $\mathfrak{A}_{\emptyset}^{eq}$ from the $=$*Subst* example, since recursively matching terms is generally a harder task than retrieving a line containing an equation. The overall rating of a command agent is set to the average rating of its single argument agents.

Although this rating system increased the effectiveness of the command suggestions, it is very inflexible as ratings are assigned by the programmer of a particular agent only. It is neither designed nor intended for being adjusted by the user at runtime as steadily controlling, e.g., more than 500 agents (this amount is easily reached by an interactive prover with only 50 tactics and an average of 10 agents per associated command) would rather divert the users attention from his main intention, namely interactively proving theorems. Anyway, since choosing an appropriate complexity rating depends on run-time and computational performance, i.e. it is on a sub-symbolic level, the user should be as far as possible spared from this kind of fine-tuning of the mechanism.

5 Resource Adaptive Approach

In this section we extend the resource adapted approach into a resource adaptive one. While we retain the principle of activation/deactivation by comparing the particular complexity ratings of the argument agents with the overall deactivation threshold, we now allow the individual complexity ratings of argument agents to be dynamically adjusted by the system itself. Furthermore, we introduce a special classification agent which analyzes and classifies the current proof goal in order to deactivate those agents which are not appropriate with respect to the current goal.

5.1 Dynamic Adjustment of Ratings

The dynamic adjustment takes place on both layers: On the bottom layer we allow the argument agents to adjust their own ratings by reflecting their performance and contributions in the past. On the other hand the command agents on the top layer adjust the ratings of their associated argument agents. This is

motivated by the fact that on this layer it is possible to compare the performance and contribution of agent societies of the bottom layer.

Therefore, agents need an explicit concept of resources enabling them to communicate and reason about their performance. The communication is achieved by propagating resource informations from the bottom to the top layer and vice versa via the blackboards. The actual information is gathered by the agents on the bottom layer of the architecture. Currently the argument agents evaluate their effectiveness with respect to the following two measures:

1. the absolute cpu time the agents consume, and
2. 'the patience of the user', before executing the next command.

(1.) is an objective measure that is computed by each agent at runtime. Agents then use these values to compute the average cpu time for the last n runs and convert the result into a corresponding complexity rating.

Measure (2.) is rather subjective which expresses formally the ability of an agent to judge whether it ever makes contributions for the command suggesting process in the current proof state. Whenever an agent returns from a computation without any new contribution to the suggestion blackboard, or even worse, whenever an agent does not return before the user executes another command (which reinitializes the blackboards), the agent receives a penalty that increases its complexity rating. Consequently, when an agent fails to contribute several times in a row, its complexity rating quickly exceeds the deactivation threshold and the agent retires.

Whenever an argument agent updates its complexity rating this adjustment is reported to the corresponding command agent via a blackboard entry. The command agent collects all these entries, computes the average complexity rating of his argument agents, and reports the complete resource information on his society of argument agents to the command blackboard. The command blackboard therefore steadily provides information on the effectiveness of all active argument agents, as well as information on the retired agents and an estimation of the overall effectiveness every argument agent society.

An additional *resource agent* uses this resource information in order to reason about a possibly optimal resource adjustment for the overall system, taking the following criteria into account:

- Assessment of absolute cpu times.
- A minimum number of argument agents should always be active. If the number of active agents drops below this value the global complexity value is readjusted in order to reactivate some of the retired agents.
- Agent societies with a very high average complexity rating and many retired argument agents should get a new chance to improve their effectiveness. Therefore the complexity ratings of the retired agents is lowered beneath the deactivation threshold.
- In special proof states some command agent (together with their argument agents) are excluded. For example, if a focused subproblem is a propositional logic problem, commands invoking tactics dealing with quantifiers are needless.

Results from the resource agent are propagated down in the agent society and gain precedence over the local resource adjustments of the single agents.

5.2 Informed Activation & Deactivation

Most tactics in an interactive theorem prover are implicitly associated with a specific logic (e.g., propositional, first-order, or higher -order logic) or even with a specific mathematical theory (e.g., natural numbers, set theory). This obviously also holds for the proof problems examined in a mathematical context. Some systems – for instance the ΩMEGA-System – do even explicitly maintain respective knowledge by administering all rules, tactics, etc., as well as all proof problems within a hierarchically structured theory database. This kind of classification knowledge can fruitfully be employed by our agent mechanism to activate appropriate agents and especially to deactivate non-appropriate ones. Even if a given proof problem cannot be associated with a very restrictive class (e.g., propositional logic) from the start, some of the subproblems subsequently generated during the proof probably can. This can be nicely illustrated with our example: The original proof problem belonging to higher-order logic gets transformed by the backward application of $\Rightarrow_I$, $=_{\text{Subst}}$ and $\equiv 2=$ into a very simple propositional logic problem (cf. line L_4). In this situation agents associated with a command from first- or higher-order logic (like $=Subst$, $\forall E$, or LEO[4]) should be disabled, whereas other agents could use this information in order to suggest the application of an automatic theorem prover that can efficiently deal with propositional logic. In the case of our example the mechanism would subsequently suggest to apply OTTER on the remaining problem.

Therefore, we add a classification agent to our suggestion mechanism whose only task is to investigate each new subgoal in order to classify it with respect to the known theories or logics. As soon as this agent is able to associate the current goal with a known class or theory it places an appropriate entry on the command blackboard (cf. "HO" entry in Fig. 1). This entry is then broadcasted to the lower layer suggestion blackboards by the command agents where it becomes available to all argument agents. Each argument agent can now compare its own classification knowledge with the particular entry on the suggestion blackboard and decide whether it should perform further computations within the current system state or not.

The motivation for designing the subgoal classifying component as an agent itself is clear: It can be very costly to examine whether a given subgoal belongs to a specific theory or logic. Therefore this task should be performed concurrently by the suggestion mechanism and not within each initialization phase of the blackboard mechanism. Whereas our current architecture provides one single classification agent only, the single algorithms and tests employed by this component can generally be further distributed by using a whole society of classification agents.

[4] A higher-order theorem prover integrated in ΩMEGA.

By appropriately extending the message passing and communication facilities of the agents, respectively the blackboards, it will even be possible to pass control knowledge (e.g., abstract knowledge on the recent proof attempt as a whole) from ΩMEGA's conceptual proof planning layer successively to the parameter and command agent layer. This extension of our mechanism may then be employed by ΩMEGA's proof planner itself to concurrently compute all applicable proof methods (with all possible instantiations) with respect to the available control knowledge instead of using traditional sequential techniques for this purpose.

6 Conclusion and Future Work

In this paper we reported on the extension of the concurrent command suggestion mechanism [4] to a resource adaptive approach. The resources that influence the performance of our system are:

(i) The available computation time and memory space.
(ii) Classification knowledge on the single agents and the agent societies.
(iii) Criteria and algorithms available to the classification agent.

Our approach can be considered as an instance of a boundedly rational system [20, 19]. The work is also related to [12] which presents an abstract resource concept for multi-layered agent architectures. [15] describes a successful application of this framework within the Robocup simulation. Consequently some future work should include a closer comparison of our mechanism with this work.

The idea to use parallel processing within automated deduction dates back to the seventies [17]. Recent work is mainly on frameworks for concurrent or cooperating automated theorem provers [9, 8, 7]. These frameworks generally involve only a very limited number of single reasoning agents. However, there are already some attempts to introduce full-scale multi-agent technology within proof planning [10].

In contrast, our work is, at least initially, not designed for realizing agent-based automated theorem proving but for assisting the user in interactive theorem proving. Our suggestion mechanism enhances traditional user support with the following features:

Flexibility The user can choose freely among several given suggestions and can even communicate with the system about parameter instantiations (by pre-specifying particular instantiations as constraints for the mechanism).

Anytime character At any given point the command blackboard contains the heuristically best-rated suggestions with respect to state of the parameter agents' computations.

Robustness Single faulty agent specifications have only a minor influence on the quality of the overall suggestion mechanism and in contrast to a traditional sequential mechanisms semi-decidable specification criteria can be employed.

Expandability The sketched mechanism is not restricted to rules and tactics and can be applied to arbitrary commands (e.g., to support an *intelligent* flag-setting for external/internal reasoners with respect to the current proof state).

User adaptability Expert-users may define their own suggestions agents. It should be possible to extend the approach such that it takes particular user preferences for certain commands (types of proofs) into account.

The presented extensions are currently implemented and analyzed in ΩMEGA. This might yield further possible refinements of the resource concepts to improve the performance of the mechanism. Another question in this context is, whether learning techniques can support our resource adjustments on the top layer, as it seems to be reasonable that there even exist appropriate resource patterns for the argument agents in dependence of the focused subproblem. We speculate that the presented mechanism can be further extended to form an intelligent, resource- and user-adaptive partner for the user in an interactive theorem prover.

Another application we are currently investigating is the use of the agent mechanism within a proof planning scenario as introduced in [6]. Since the approach is not restricted to specify applicability conditions for rules and tactics only, it can analogously be employed for proof methods as well. So far the specification of the argument agents for rules and tactics describes structural properties of single arguments as well as structural dependencies between the different arguments. Similarly we can specify agents which check, probably guided by available control knowledge, the particular pre-conditions of a proof method, i.e. check for proof lines matching with those required for a method to be applicable or verify additional application conditions of the method. Thus the hope is that we can at least to some extend exploit concurrency even within a traditional proof planner when computing the applicable methods with respect to the given proof goal by cooperating argument agents in each proof step [5].

Acknowledgments We would like to thank Serge Autexier and Christoph Jung for stimulating discussions.

References

1. P. B. Andrews. *An Introduction To Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, San Diego, CA, USA, 1986.
2. 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.
3. 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)*, LNAI, Townsville, Australia, 1997. Springer Verlag, Berlin, Germany.

4. 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.
5. Christoph Benzmüller and Volker Sorge. Towards Fine-Grained Proof Planning with Critical Agents. In Manfred Kerber, editor, *Informal Proceedings of the Sixth Workshop on Automated Reasoning Bridging the Gap between Theory and Practice in conjunction with AISB'99 Convention*, pages 20–22, Edinburgh, Scotland, 8–9 April 1999. extended abstract.
6. 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.
7. Jörg Denzinger and Ingo Dahn. Cooperating theorem provers. In Wolfgang Bibel and Peter Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume 2, pages 483–416. Kluwer, 1998.
8. Jörg Denzinger Dirk Fuchs. Knowledge-Based Cooperation between Theorem Provers by TECHS. SEKI REPORT SR-97-11, Fachbereich Informatik, Universität Kaiserslautern, 1997.
9. 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.
10. Michael Fisher and Andrew Ireland. Multi-agent proof-planning. In *Workshop on Using AI Methods in Deduction at CADE-15*, July 6–9 1998.
11. G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.
12. C. Gerber and C. G. Jung. Resource management for boundedly optimal agent societies. In *Proceedings of the ECAI'98 Workshop on Monitoring and Control of Real-Time Intelligent Systems*, pages 23–28, 1998.
13. M. J. Gordon, R. Milner, and Ch. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *LNCS*. Springer Verlag, Berlin, Germany, 1979.
14. M. J. C. Gordon and T. F. Melham. *Introduction to HOL*. Cambridge University Press, Cambridge, United Kingdom, 1993.
15. C. G. Jung. Experimenting with layered, resource-adapting agents in the robocup simulation. In *Proc. of the ROBOCUP'98 Workshop*, 1998.
16. William McCune and Larry Wos. Otter CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.
17. Stuart C. Shapiro. Compiling Deduction Rules from a Semantic Network into a Set of Process. In *Abstracts of Workshop on Automated Deduction*, Cambridge, MA, USA, 1977. MIT. Abstract only.
18. J. Siekmann, S. M. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, M. Kohlhase, K. Konrad, E. Melis, A. Meier, and V. Sorge. $\mathcal{LOUI}$: A Distributed Graphical User Interface for the Interactive Proof System ΩMEGA. Submitted to the International Workshop on User Interfaces for Theorem Provers, 1998.
19. H. A. Simon. *Models of Bounded Rationality*. MIT Press, Cambridge, 1982.
20. S. Zilberstein. Models of Bounded Rationality. In *AAAI Fall Symposium on Rational Agency*, Cambridge, Massachusetts, November 1995.