A Blackboard Architecture for Guiding Interactive Proofs

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. The acceptance and usability of current interactive theorem proving environments is, among other things, strongly influenced by the availability of an intelligent default suggestion mechanism for commands. Such mechanisms support the user by decreasing the necessary interactions during the proof construction. Although many systems offer such facilities, they are often limited in their functionality. In this paper we present a new agent-based mechanism that independently observes the proof state, steadily computes suggestions on how to further construct the proof, and communicates these suggestions to the user via a graphical user interface. We furthermore introduce a focus technique in order to restrict the search space when deriving default suggestions. Although the agents we discuss in this paper are rather simple from a computational viewpoint, we indicate how the presented approach can be extended in order to increase its deductive power.

1 Introduction

Interactive theorem provers have been developed in the past to overcome the shortcomings of purely automatic systems by enabling the user to guide the proof search and by directly importing expert knowledge into the system. For large proofs, however, this task might become difficult when the system does not support the user's orientation within the proof and does not provide a sophisticated suggestion mechanism in order to minimize the necessary interactions.

Current interactive theorem proving systems such as ΩMEGA [3] or TPS [2] already provide mechanisms for suggesting command arguments or even commands. Usually these mechanisms are rather limited in their functionality as they (i) use a sequential strategy allowing only for argument suggestions in a particular order, and (ii) only work in interaction with the user and do not autonomously search for additional suggestions in a background process.

In this paper we suggest an agent-based approach, which separates the default suggestion mechanism in most parts from the interactive theorem proving process. For that we use concurrent programming techniques by introducing autonomous agents which self-containedly search for default suggestions and which cooperate by exchanging relevant results via a Blackboard architecture [4]. This

F. Giunchiglia (Ed.): AIMSA'98, LNAI 1480, pp. 102–114, 1998.
© Springer-Verlag Berlin Heidelberg 1998

mechanism is steadily working in the background and dynamically adapting its computational behavior to the state of the proof. It is able to exchange command and argument suggestions with the user via a graphical user interface. We furthermore introduce a focus mechanism which explicitly partitions a partial proof into subproblems in order to restrict the search process of the default mechanism.

The presented approach is an improvement of traditional mechanisms for computing command defaults in interactive theorem provers, as it on one hand enhances the quality of suggestions by special guidance for computation of default values. On the other hand it offers the user the choice between several suggestions that are computed in parallel and sorted according to goal-directed heuristics. Furthermore, in our approach the system steadily conducts useful computations in the background and constantly utilizes the available computational resources.

The paper is organized in seven sections. Section 2 briefly introduces some fundamental concepts. In Sec. 3 we present the basic mechanism for computing argument suggestions. Section 4 describes a technique for structuring proofs that guides the suggestion mechanism. Section 5 extends this mechanism to an architecture for suggesting commands. Further extensions of our approach are sketched in Sec. 6. Finally, Sec. 7 discusses some related work and summarizes our conclusions.

2 Preliminaries

In this section we give brief definitions of some concepts we will refer to in the remainder of this paper. We assume that the reader is familiar with the natural deduction calculus (ND) [7]. In *Ω*MEGA [3] as well as in many other interactive theorem proving systems, such as TPS [2] or HOL [8], theorems are interactively proven by applying ND-rules or tactics, where the latter are essentially compounds of ND-rules. In other words, a tactic application abbreviates a purely rule based derivation, whereby the size of this derivation, called **expansion size**, generally depends on both, the specific definition of the tactic and the particular arguments within the application. For instance, an application of the tactic $\forall_E^*$ (see specification below) on a formula with two universally quantified variables expands to a twofold consecutive application of rule $\forall_E$ and has expansion size 2.

Since a rule can be regarded as a trivial instance of a tactic, we will only use the notion **tactic**. The following five tactics are typical representatives which we will refer to in the remainder of this paper.

$$\frac{\forall x.A}{[t/x]A}\ \forall_E(t) \qquad \frac{\forall x_1,\ldots,x_n.A}{[t_1/x_1,\ldots,t_n/x_n]A}\ \forall_E^*(t_1,\ldots,t_n) \qquad \frac{A \quad B}{A \wedge B}\ \wedge_I$$

$$\frac{A \vee B \quad \begin{matrix}[A]\\ \vdots \\ C\end{matrix} \quad \begin{matrix}[B]\\ \vdots \\ C\end{matrix}}{C}\ \vee_E \qquad \frac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A \Rightarrow B}\ \Rightarrow_I$$

Tactics allow for the manipulation of **partial proof trees** consisting of sets of **proof lines**. These sets contain **open lines** which are subgoals yet to be proven, and **support lines** which are either hypotheses or lines derived from these by means of tactic applications. A partial proof tree is said to be **closed**, when it no longer contains any open lines.

In the ΩMEGA-system tactics have a fixed number of premises $(p_1, \ldots, p_l)$ and conclusions $(c_1, \ldots, c_m)$. They may furthermore require some additional arguments $(t_1 \cdots t_n)$, e.g. the term t in tactic $\forall_E$. All possible arguments of a tactic are specified by its **argument pattern** which has the general form $\frac{p_1 \cdots p_l}{c_1 \cdots c_m}\ r(t_1 \cdots t_n)$. For the execution of a command some elements of the argument pattern have to be instantiated. Such a **partial argument instantiation (PAI)** refers to an arbitrary constellation of given and missing command arguments (premises, conclusions and additional arguments). Note that a tactic can usually be applied in more than one direction. As an example we consider the $\wedge_I$ tactic that can be applied in five different directions: backward - where $A \wedge B$ is the given goal and A and B are computed; forward - where the conclusion is computed from the given premises; two sideway directions - when the conjunction and either one of the conjuncts are provided and the remaining conjunct is computed; and finally for closing the conclusion when all three arguments are specified. Thus we have five possible PAIs for a single argument pattern.

3 Computing Defaults

Rules and tactics usually provide implicit information relating its particular arguments to each other. This dependencies can (and should be) used in an intelligent system in order to compute **command argument suggestions** depending on the proof state and some already specified argument instantiations. As an example we consider the application of the tactic $\forall_E$ with the corresponding command pattern $\frac{p}{c}\ \forall_E(t)$ in the partial proof on the right[1]. The structural information for the single arguments of this tactic is that p has to be instantiated with a universally quantified formula and that the formula instantiating the conclusion c should match with the scope of the universally quantified formula for p, whereas the actual matcher is chosen as instance for t.

A_1	(A_1)	$\vdash \forall y. R(y)$	Hyp
A_2	(A_2)	$\vdash \forall z. Q(z)$	Hyp
Conc	$(A_1 A_2)$	$\vdash R(u)$	OPEN

Argument predicates and functions In order to model the dependencies between the different arguments we use **argument predicates** and **argument functions**. While the former denotes conditions the formula of a proof line has to

[1] The partial proof is displayed in a linearized representation of the ND-calculus as described in [1]. Each line consists of a unique label, a set of hypotheses, a formula, and a justification.

$$
\begin{aligned}
p : & \mathfrak{p}^p_\emptyset = [\lambda x{:}p.\, \text{is-universal-quantification}(x)]_\emptyset \\
& \mathfrak{p}^p_{\{c\}} = [\lambda x{:}p.\, \text{matches}(\text{scope}(x), u)]_{\{u:c\}} \\
& \mathfrak{p}^p_{\{c,t\}} = [\lambda x{:}p.\, \text{applicable}(\forall_E, x, u, v)]_{\{u:c, v:t\}} \\
c : & \mathfrak{p}^c_{\{p\}} = [\lambda x{:}c.\, \text{matches}(\text{scope}(u), x)]_{\{u:p\}} \\
& \mathfrak{p}^c_{\{p,t\}} = [\lambda x{:}c.\, \text{applicable}(\forall_E, u, v, x)]_{\{u:p, v:t\}} \\
t : & \mathfrak{f}^t_{\{p,c\}} = [\text{let subst } = \text{match}(\text{scope}(u), c) \text{ in if} \\
& \#\text{dom}(\text{subst}) > 1 \text{ then } \bot \text{ else dom}(\text{subst})]_{\{u:p, v:c\}}
\end{aligned}
$$

fulfill, the latter contains algorithms to compute an additional argument with respect to some given arguments. The table on the right presents five argument predicates, three for the argument p and another two for c, and one argument function for t. We use a λ-notation to specify the particular predicates. As further notational conventions we have that $\mathfrak{p}$ and $\mathfrak{f}$ denote predicates and functions respectively. The subscript sets specify the assumed degree of instantiation of the command pattern — we refer to its elements as **necessary arguments** — and the superscript symbol refers to the argument in the command pattern the predicate (or function) is associated with. Instances for the necessary arguments are mapped to their counterparts in the command pattern by the : operator.

We observe this with the example of the $\mathfrak{p}^p_{\{c\}}$ predicate, assuming that the c has been instantiated with the line *Conc* from the above examples. Matching the formula $R(u)$ with the respective scopes of the support lines of *Conc* yields a positive result for line A_1, which is subsequently bound to the argument p. With both p and c instantiated, the argument function $\mathfrak{f}^t_{\{p,c\}}$ can be employed. The result of its computations is the domain of $y \leftarrow u$, which is the matcher of $R(y)$ and $R(u)$. Thus, u is suggested as instance for the additional argument t.

In case the computation of $\mathfrak{f}^t_{\{p,c\}}$ is not successful, it returns an unspecified argument, indicated by $\bot$. This possibility may arise in case some of the necessary arguments are specified by the user. $\bot$ is also automatically instantiated as default if no argument predicate or function is applicable or an argument predicate fails to retrieve a valid proof line. In the specification of the argument predicates and functions of $\forall_E$ we have not considered all possible combinations of necessary arguments but only those that made sense in the context of the tactic.

Argument Agents For each argument predicate $\mathfrak{p}^a_L$ and argument function $\mathfrak{f}^{a'}_{L'}$ the system automatically creates **Predicate Agents** $\mathfrak{A}^a_L$ and **Function Agents** $\mathfrak{A}^{a'}_{L'}$ respectively. We use **Argument Agents** as generic name for both agent types.

The intention of the Predicate Agents is to search for proof lines which fulfill the underlying constraints. Predicate Agents searching for premise arguments restrict their search to the support lines of the given partial proof, while those searching for conclusion arguments only consider open lines. Function Agents, on the other hand, just compute their suggestions. An Argument Agent gets active as soon as all elements of the set of necessary arguments are provided. For example the agent $\mathfrak{A}^p_\emptyset$ starts its search immediately when the command for $\forall_E$ is chosen by the user, whereas the agent $\mathfrak{A}^c_{\{p,t\}}$ has to wait until instantiations

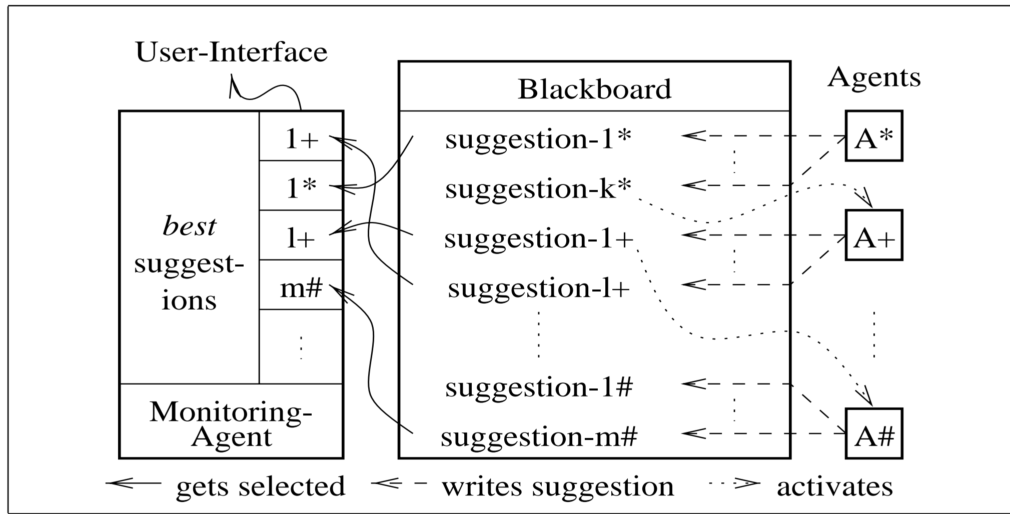


Fig. 1. The argument suggestion mechanism

for p and t are provided. The communication between agents is organized using a Blackboard architecture.

Suggestion Blackboard Results returned by the Argument Agents are written to a **Suggestion Blackboard**. Each agent adds a newly discovered argument suggestion to the set of its necessary arguments and thus the entries on the Blackboard are PAIs. PAIs correspond also to the sets of necessary arguments that trigger other Argument Agents. As example we give the Blackboard entry $\{A_1{:}p,\ Conc{:}c\}$ that activates the Agent $\mathfrak{A}^t_{\{p,c\}}$ in the above proof. Then $\mathfrak{A}^t_{\{p,c\}}$ returns as result $\{A_1{:}p,\ Conc{:}c,\ u{:}t\}$.

Computing suggestions We now sketch how the particular agents interact via the entries on the Blackboard and how the user can interact with the suggestion mechanism via ΩMEGA's graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [11]. Suppose a user chooses to apply $\forall_E$ in our example. The choice of the command automatically initializes a corresponding empty Suggestion Blackboard. As only $\mathfrak{A}^p_\emptyset$ needs no necessary arguments, it will be the sole Argument Agent becoming active immediately. We assume that the first result of $\mathfrak{A}^p_\emptyset$ is $\{A_2{:}p\}$. With this as Blackboard entry, the agent $\mathfrak{A}^c_{\{p\}}$ is activated which, as $R(u)$ and $Q(z)$ do not match, fails to find an appropriate open line. However, in the meantime $\mathfrak{A}^p_\emptyset$ has returned $\{A_1{:}p\}$, another result that $\mathfrak{A}^c_{\{p\}}$ can use. While $\mathfrak{A}^p_\emptyset$ stops its search, since there are no other support lines to consider, $\mathfrak{A}^c_{\{p\}}$ returns $\{Conc{:}c,\ A_1{:}p\}$, thereby triggering $\mathfrak{A}^t_{\{p,c\}}$ as described earlier. When all agents have terminated their search, we have four entries on the Suggestion Blackboard: $\{A_2{:}p\}$, $\{A_1{:}p\}$, $\{Conc{:}c,\ A_1{:}p\}$ and $\{A_1{:}p,\ Conc{:}c,\ u{:}t\}$.

In order to immediately provide the user with argument suggestions without waiting until all Argument Agents have terminated their search, the Suggestion

Blackboard is constantly observed. This is done by a Monitoring Agent, which schematically works as displayed in Fig. 1. The agent monitors the Suggestion Blackboard and, as soon as entries appear, offers the respective PAIs as choices to the user. These choices are sorted according to criteria which we elaborate in Sec. 4.

Naturally the user has the possibility to interfere with the suggestion process by simply entering arguments for the selected command. Suppose a user would enter the line *Conc* as desired conclusion line for $\forall_E$, then the Argument Agent $\mathfrak{A}^p_{\{c\}}$ would become active, suggesting A_1 as instantiation of argument p. This again triggers $\mathfrak{A}^t_{\{p,c\}}$ and eventually completes the PAI. Each working agent terminates as soon as it realizes that the Suggestion Blackboard it works for is no longer available, which is generally the case when the command the Blackboard is associated with is executed.

The interactive suggestion approach can only be comfortably used in connection with a graphical user interface like $\mathcal{L}\Omega\mathcal{UI}$ [11], where the user can specify arguments in a command mask by clicking with the mouse to proof lines and demand suggestions in no particular order. The disadvantage of a command shell interface usually is that it allows to specify the arguments in a fixed order only, so that an arbitrary partial argument selection can hardly be implemented.

4 Foci

So far the Predicate Agents introduced in the preceding section search blindly either among all open lines or all support lines of the proof. The fact that the logical structure of the proofs is not considered can lead to major errors in the suggested arguments, as a set of arguments might be proposed that is logically incompatible. Therefore we elaborate in this section a focus technique that suitably guides the search for default suggestions by restricting it to certain subproblems. It explicitly keeps track of the implicitly given structural and chronological information given in a partial proof and enables the reuse of this information in case some already justified proof lines are deleted and the system backtracks to a previous proof state.

An example is given in Fig. 2 which presents a partially linearized natural deduction proof on the left together with the explicit graphical representation of its proof structure on the right. The proof problem consists in showing that theorem T: $((C \Rightarrow D) \Rightarrow (A \Rightarrow D)) \wedge A$ can be derived from the assumptions A_1: $A \wedge B$ and A_2: $C \vee D$. By applying the $\wedge_I$-rule backwards to T the lines L_1 and L_2 are introduced and the proof problem is partitioned into two subproblems. The next two proof steps – two backward applications of $\Rightarrow_I$ – solely belong to the first subproblem, i.e. to derive L_1 from A_1 and A_2. Thus instead of always searching for argument suggestions among all proof lines we restrict our mechanism to the nodes belonging to the focused subproblem – called the active focus – and furthermore guide the search with respect to the chronological dependencies of those node.

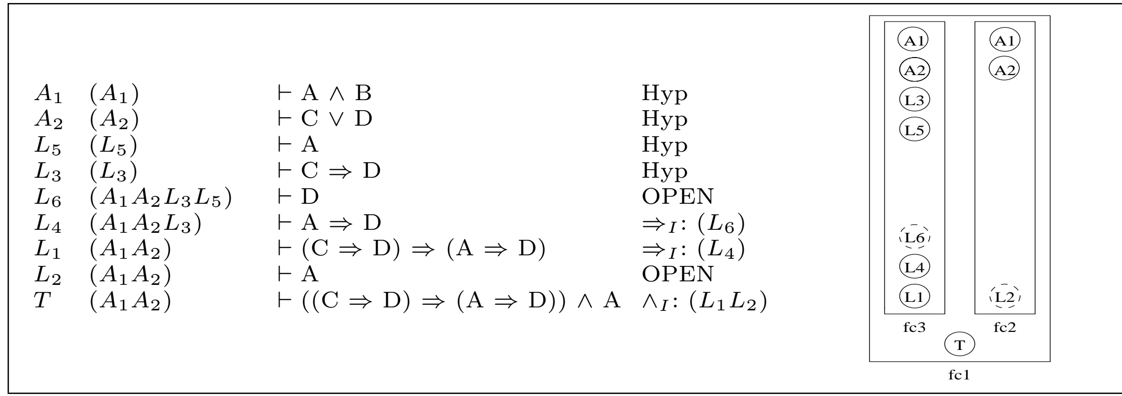
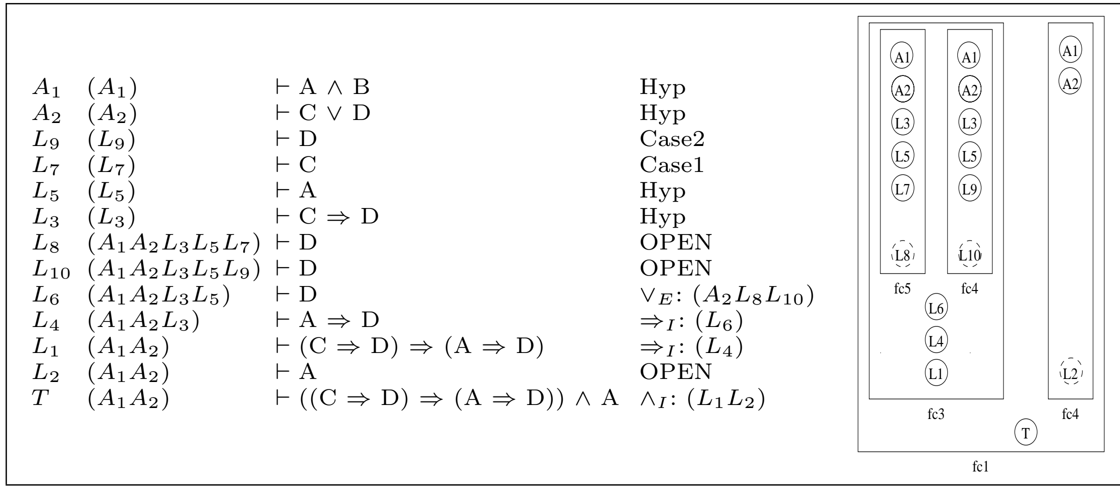


Fig. 2. Proof context pc_1

Definitions We will give the definitions of the main concepts involved and point out their counterpart in Fig. 2. Given a partial proof P consisting of the set of its proof lines L_P, we define:

- chronological line-order $<_l$: We assume the existence of a total order $<_l : L_P \times L_P$ on the proof lines of P such that for all $l_1, l_2 \in L_P$ holds: $l_1 <_l l_2$, iff proof line l_1 was inserted in P before l_2. We call such an order a **chronological order for** P.
- focus: Let $l \in L_P$ and $\mathcal{SL}, \mathcal{DL} \subseteq L_P$. We call the triple $f = (\mathcal{SL}_l, l, \mathcal{DL}_l)$ a **focus**, if $\mathcal{SL}_l$ is a set of support lines of l and $\mathcal{DL}_l$ a set of descendent lines of l. l is called the **focussed line** of f. If l is an open line, we call f **open**, otherwise **closed**. The set of all foci of P is denoted by F_P. Two examples of foci are displayed in Fig. 2 with the boxes fc_2 and fc_3. There, the elements of the sets of support lines, displayed in the top of the boxes, are ordered with respect to $<_l$. Intuitively foci describe subproblems of a partial proof which can be solved independent from the other parts of the proof.
- focus context: Using foci as base constructions we inductively define the set FC_P of **focus contexts** of P as the smallest set containing:
  - (i) $F_P \subseteq FC_P$
  - (ii) Let $fc_1, \ldots, fc_n \in FC_P$ be a set of focus contexts with respective sets of derived lines $\mathcal{DL}_1 \ldots \mathcal{DL}_n$. Let k be a line in P with premises $k_1, \ldots, k_n$, where $k_i \in \mathcal{DL}_i$ for $1 \leq n$, and let $\mathcal{DL}_k$ be the set of descendent lines of k. Then the triple $fc = ((fc_1, \ldots, fc_n), k, \mathcal{DL}_k)$ is called a **focus context** of P with $fc \in FC_P$.

  An example for a focus context is given by fc_1 in Fig. 2. Focus contexts structurally partition a proof problem into certain subproblems.
- foci priority order $\prec$: Given a set $S \subseteq FC_P$ of focus contexts of P. A total ordering $\prec$: $S \times S$ is called a **foci priority order** on S.
- proof context: Let $S \subseteq FC_P$ be a set of focus contexts of P, $<_l$ be a chronological line-order for P and $\prec$ a foci priority order on S. We then call the triple $pc = (S, <_l, \prec)$ a **proof context** for P. Note that for each focus context fc in S the restriction of $<_l$ on the set of support lines of fc is unique.



A_1	(A_1)	$\vdash$ A $\wedge$ B	Hyp
A_2	(A_2)	$\vdash$ C $\vee$ D	Hyp
L_9	(L_9)	$\vdash$ D	Case2
L_7	(L_7)	$\vdash$ C	Case1
L_5	(L_5)	$\vdash$ A	Hyp
L_3	(L_3)	$\vdash$ C $\Rightarrow$ D	Hyp
L_8	$(A_1 A_2 L_3 L_5 L_7)$	$\vdash$ D	OPEN
L_{10}	$(A_1 A_2 L_3 L_5 L_9)$	$\vdash$ D	OPEN
L_6	$(A_1 A_2 L_3 L_5)$	$\vdash$ D	$\vee_E$: $(A_2 L_8 L_{10})$
L_4	$(A_1 A_2 L_3)$	$\vdash$ A $\Rightarrow$ D	$\Rightarrow_I$: (L_6)
L_1	$(A_1 A_2)$	$\vdash$ (C $\Rightarrow$ D) $\Rightarrow$ (A $\Rightarrow$ D)	$\Rightarrow_I$: (L_4)
L_2	$(A_1 A_2)$	$\vdash$ A	OPEN
T	$(A_1 A_2)$	$\vdash$ ((C $\Rightarrow$ D) $\Rightarrow$ (A $\Rightarrow$ D)) $\wedge$ A	$\wedge_I$: $(L_1 L_2)$

Fig. 3. Proof context pc_2

An example for a proof context is $pc_1 = (\{fc_1, fc_2, fc_3\}, <_l, \prec)$ in Fig. 2 with $T <_l A_1 <_l A_2 <_l L_1 <_l \ldots <_l L_6$ and $fc_1 \prec fc_2 \prec fc_3$.

· active focus: Given a proof context $pc = (S, <_l, \prec)$. Then we call the uniquely defined open focus context $fc \in S$ that is maximal with respect to $\prec$ the *active focus context* of pc.

As already mentioned, the idea of the above concepts is to restrict the search of our Argument Agents to the lines of the active focus and furthermore to guide the search process on the specified line-order.

Reasoning inside proof contexts We omit the formal definition of transitions of proof contexts. Instead we exemplify how the above focus mechanism is being initialized and updated during the construction of a proof by further elaborating the example in the Figs. 2 and 3. Initially, our partial proof P consists of one open line T and its hypotheses A_1, A_2, forming the proof context $pc_0 = (fc_1, <_l, \prec)$ where $T <_l A_1 <_l A_2$. Here the order of the hypotheses is chosen randomly.

Generally, in each proof state we can apply tactics (i) forward to the support lines, (ii) backward to the open line, or (iii) fully instantiated in order to close a subproblem. We will exemplify how a proof context is dynamically updated in all of these cases by considering the proof context $pc_1 = (\{fc_1, fc_2, fc_3\}, <_l, \prec)$ from Fig. 2.

Case (i) is rather simple as the application of a tactic to one or several support lines only introduces a new line into the respective focus context. Furthermore $<_l$ is updated so the newly introduced line becomes the maximum element.

Case (iii) just closes an active focus context. If we for instance explicitly choose fc_2 in pc_1 to become the active focus – i.e. $\prec$ is updated such that fc_2 becomes the new maximum element – and then eliminate the conjunction in A_1 in order to close L_2 along with fc_1. Thereby fc_3 would automatically become the new active focus.

Case (ii) is slightly more complicated because the structure of the proof context changes. As an example of this case we consider the transition of the old proof context $pc_1 = (\{fc_1, fc_2, fc_3\}, <_l, \prec)$, illustrated in Fig. 2, to the new context $pc_2 = (\{fc_1, fc_2, fc_3, fc_4, fc_5\}, <_l', \prec')$ given in Fig. 3. Within the active focus fc_3 of pc_1 we apply the tactic $\vee_E$ backwards to the focussed line L_6 and the disjunction line A_2. Thereby lines L_8 and L_{10} and the associated new support lines L_7 and L_9 are inserted. Most importantly, two new focus contexts, fc_4 and fc_5, are created. The former containing the nodes L_7 and L_8 and the latter L_9 and L_{10}. Furthermore, both new focus contexts inherit the set of support lines $\{A_1, A_2, L_3, L_5\}$ from their predecessor fc_3. The chronological function $<_l'$ is the obvious extension of $<_l$ and $\prec$ is expanded to $\prec'$ with $fc_1 \prec' \ldots \prec' fc_5$, where fc_5 is the new active focus context.

Guiding the Default Mechanism We now observe the behavior of the Argument Agents inside proof contexts. As there is always exactly one active focus context $fc = (\mathcal{SL}, l, \mathcal{DL})$, the Argument Agents searching for premise lines will restrict their search to the support lines in $\mathcal{SL}$ and, moreover, lines with a higher line-order are considered first. Argument Agents searching for conclusion lines of a tactic have essentially just the focussed line to consider, and can either accept or discard it as a possible suggestion.

In case a tactic has multiple conclusions, the associated agents searching for conclusion lines may search among all focussed lines of all open focus contexts. As we are interested in suggestions related to the active focus, we can filter out all those which do not contain the focussed line of the active focus. Note that searching among all open lines is of course prone to error, as there is no guarantee that a thereby acquired open line has the necessary support lines for the tactic to be applicable. This problem needs to be further examined in order to be solved in a more elegant way.

Before the set of suggestions computed by the monitoring agent is presented to the user, it is sorted according to two criteria: (1) argument combinations including the focussed line and (2) the suggestions containing lines with higher line-order are preferred. Criterion (2) can be guaranteed by sorting the suggestions with respect to a multi-set extension of $<_l$. We believe to at least partially anticipate human problem solving behavior by assuming that proof steps are probably derived when they are needed to contribute to the solution. Moreover a user is likely to prove one subgoal before considering the next. Therefore a proof is ideally constructed, with respect to the proof contexts, i.e. the focussed line in the focus context with the highest priority number is always considered first. This restriction is, of course, not always desirable and we have included the option for a user to interactively change the active focus structure by specifying a different subgoal to concentrate on. The thereby chosen focus structure is promoted by simply setting its priority number one larger than the maximum of all existing priority numbers.

5 Suggesting Commands

In this section we extend the suggestion mechanism to precompute tactics (actually commands that execute tactics) that might be applicable in the next proof step. We use both the suggestion mechanism for default values and the guiding of the focus contexts from the preceding sections. In fact, the presented method is based on a similar Blackboard architecture as the one presented in Sec. 3, and thus we will sketch only the main ideas with respect to the concepts introduced there.

In general we have a single agent and a Suggestion Blackboard associated with every command that corresponds to a tactic. The particular Suggestion Blackboards are initialized and the **Command Agents**[2] are automatically freshly started after each alteration of the partial proof. They in turn initialize and monitor Suggestion Blackboards for the respective commands which the Argument Agents can use for their computations. The intention of a Command Agent is to maintain PAI entries on the **Command Blackboard** corresponding to the best argument suggestion on its Suggestion Blackboard. The Command Blackboard itself is monitored and the results are accumulated, sorted, and presented to the user. This is done when either a fully specified PAI entry for some command has been found or after a certain time limit expires. As neither the Command Agents nor their corresponding Argument Agents will cease to work before the user executes a command which changes the partial proof, better suggestions, if there are any, can be successively incorporated on request.

As theorem provers of the size of *Ω*MEGA often contain a large number of tactics, the number of Command and Argument Agents might amount to an intractable size. Therefore, we constrain the suggestion mechanism by only considering the backwards application of tactics. This is achieved by entering the open line of the active focus context as default argument for all Argument Blackboards. Furthermore we suppress the search for tactics that are applicable in every proof state, such as proof by contradiction or elimination of negation.

The excluded tactics are appended in the end to the list of suggested tactics, which is sorted according to two criteria: (1) Commands with the most complete PAI will be promoted. This is the primary sorting criteria that guarantees that we prefer those suggestions which will increase the chance of closing a subgoal.

Criterion (2) is used to sort commands that are equivalent with respect to (1). Intuitively the idea is to prefer bigger proof steps to smaller ones. For that it uses as **rating** the expansion size of the tactic when applied with the suggested PAI. Surely, this rating depends on the particular definition of the tactic, which may not be optimal. Even though, we promote tactics with higher ratings, thereby speculating to achieve larger proof steps with the application of a single tactic. As an example we give the rule $\forall_E$ which has, being a rule, the expansion size 1 and the tactic $\forall_E*$ that expands to one or several applications of $\forall_E$ has rating

[2] Actually Command Agents are not autonomous agents like the Argument Agents but rather the knowledge source of a Blackboard. We will discuss this point in more detail in Sec. 7

2 when applied to a PAI containing two suggestion for the additional term-arguments. The approach can be improved by developing a more meaningful definition of the expansion size of tactics, e.g. by defining them with respect to a normal form derivation associated with a particular expansion.

6 Extensions of the Approach

The Argument Agents described so far are rather simple in a sense that they do not involve very complicated computations, although some necessary matchings can be very time-consuming. However, our approach is not restricted to simple computations and can be extended in order to incorporate more powerful deductive procedures or can even be connected to one of the external reasoners (such as automated theorem provers) integrated in the ΩMEGA-System. In an extended system where agents perform many costly computations or even incorporate undecidable procedures it will be necessary to control the system in accordance to available resources, such as time or memory consumption.

A certain resource concept already proves useful for the current approach as our experiments with static **Complexity Ratings** $r_{\mathfrak{A}}$ associated with each Argument Agent $\mathfrak{A}$ show. The idea of these ratings is to reflect the computation costs of the particular Argument Agents. For example, the ratings for agents performing matchings or unifications are relatively high. Depending on the user definable **Computation Complexity Level** r_u, only agents $\mathfrak{A}$ with $r_{\mathfrak{A}} \leq r_u$ are allowed to work. This resource adapted approach can be further developed to obtain a resource adaptive one, by introducing a dynamical allocation of Complexity Ratings (or general resource information) to the particular agents in dependence of their performance in the past.

Another use of the distribution aspect could be that the system tries to solve open subgoals in the background. While the user concentrates on the active focus and gets supported by the suggestion mechanism, a couple of powerful deductive agents (e.g. agents connected to external automated theorem provers as available in the ΩMEGA-System) concentrate on the non-active open foci and try to close them automatically. Positive attempts are signalled to the user, who can then accept or reject the suggested proofs. Those deductions can also be combined with the Command Suggestion Mechanism by repeatedly suggesting commands and applying the heuristically preferred one automatically to the proof.

7 Related Work and Conclusion

The necessity to develop (graphical) user interfaces and mechanisms to better support the user within interactive theorem proving environments has been pointed out by [13]. Even though, current theorem proving environments still offer many potentialities for an improvement and still have not adapted all the usable techniques developed for user interfaces in other domains [12]. Focusing techniques are well known from natural language processing and have been suggested as a tool for guidance in Graphical User Interfaces in [9].

In this paper we have discussed a new approach for an intelligent suggestion of commands as well as their corresponding arguments within the framework of an interactive theorem prover. Both mechanisms employ techniques from concurrent programming in order to compute the desired suggestions. We have combined this machinery with a focusing technique that can guide a user even through large proofs, as well as enhance the computation of default values for commands and arguments.

In our context focusing is done on two layers, where one layer pays tribute to the logical dependencies inside focused subproofs and the other credits the chronological order of introduced proof steps. Thereby both the suggestion mechanism is guided in a senseful way and during interactive proof construction subproblems can be concisely presented.

Our suggestion mechanism does not waste resources, as it constantly works in a background process and steadily improves its suggestions with respect to its heuristic selection criteria. Furthermore, in contrast to conventional command and argument suggestion mechanisms, our approach provides more flexibility in the sense that agents can be defined independently from the actual command and the user can choose freely among several given suggestions.

In contrast to the classical HEARSAY Blackboard architecture used for speech recognition (cf. [5,6]) our Argument Agents do not depend on an explicit central control system. Instead they can be seen as autonomous entities. Their actions are solely triggered by the type of the Blackboard, that is, the command for which the Blackboard accumulates argument suggestions and the blackboard entries themselves. Contrary to this, the presented Command Agents are not autonomous agents in the same sense as they completely depend on the respective Command Blackboard. However, by calling them agents we decided on a uniform nomination in our hierarchy of Command and Suggestion Blackboards.

Although the agents described in Secs. 3 and 5 are rather simple, we have motivated some extensions in order to increase the deductive power of our approach. We are convinced that the described Blackboard architecture can form a suitable basis for a distributed interactive theorem proving environment in which as many as possible of the available, but in traditional systems mostly unused, computation resources are sensibly used by deductive agents in order to contribute to the proof construction.

The focusing technique and the basic suggestion mechanism have been implemented in the ΩMEGA-system. But as the core of ΩMEGA, including the command mechanism, is still implemented in COMMON-LISP, we cannot take full advantage of the distributed agent approach yet. However, most of ΩMEGA's interface functionality is currently reimplemented in OZ [10], a concurrent programming language in which ΩMEGA's graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [11] is already running. This reimplementation will then make use of the full suggestion mechanism we have developed in this paper and will furthermore enable the realization of the extensions described in Sec. 6.

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. P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. 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. R. Engelmore and T. Morgan, editors. *Blackboard Systems.* Addison-Wesley, 1988.
5. L. D. Erman, F. Hayes-Roth, and R. D. Reddy. The HERSAY-II speech understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2), 1980.
6. L. Erman, P. London, and S. Fickas. The Design and an Example Use of HEARSAY-III. In P. J. Hayes, editor, *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 409–415. William Kaufmann, 1981.
7. G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.
8. M. J. C. Gordon and T. F. Melham. *Introduction to HOL.* Cambridge University Press, Cambridge, United Kingdom, 1993.
9. M. A. Pérez and J. L. Sibert. Focus in graphical user interfaces. In W. D. Gray, William E. Hefley, and Dianne Murray, editors, *Proceedings of the International Workshop on Intelligent User Interfaces*, pages 255–258. ACM Press, 1993.
10. The Oz Programming System Programming Systems Lab, DFKI, , Germany, 1998. URL: `http://www.ps.uni-sb.de/oz/`.
11. J. Siekmann, S. M. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, M. Kohlhase, K. Konrad, E. Melis, A. Meier, and V. Sorge. $\mathcal{L}\Omega\mathcal{UI}$: A Distributed Graphical User Interface for the Interactive Proof System ΩMEGA. Submitted to the International Workshop on User Interfaces for Theorem Provers, 1998.
12. J. W. Sullivan and S. W. Tyler, editors. *Intelligent User Interfaces.* ACM Press Frontier Series. ACM Press, New York, NY, USA, 1991.
13. L. Théry, Y. Bertot, and G. Kahn. Real Theorem Provers Deserve Real User-Interfaces. In *Proceedings of The Fifth ACM Symposium on Software Development Environments (SDE5)*, Washington D.C., USA, December 1992. ACM Press.