Ω-Ants – An open approach at combining Interactive and Automated Theorem Proving

Christoph Benzmüller* and Volker Sorge†
(C.E.Benzmuller@cs.bham.ac.uk, sorge@ags.uni-sb.de)

School of Computer Science, The University of Birmingham
Edgbaston, Birmingham B15 2TT, England

Fachbereich Informatik, Universität des Saarlandes,
D-66041 Saarbrücken, Germany

Abstract. *We present the Ω-Ants theorem prover that is built on top of an agent-based command suggestion mechanism. The theorem prover inherits its beneficial properties from the underlying suggestion mechanism such as run-time extendibility and resource adaptability. Moreover, it supports the distributed integration of external reasoning systems. We also discuss how the implementation and modeling of a calculus in our agent-based approach can be investigated wrt. the inheritance of properties such as completeness and soundness.*

1 Introduction

We present the new Ω-Ants automated theorem proving approach that is build on top of the Ω-Ants agent-based command suggestion mechanism. This mechanism has been originally developed to support the user in interactive theorem proving by using available resources in-between user interactions to search for the next possible proof steps [4]. This is done via a hierarchical blackboard-architecture where agents concurrently check for applicable commands (i.e. commands that apply proof rules) and the most promising commands are dynamically presented to the user. The faster a command's applicability can be analysed the faster it will be reported immediately to the user. Further benefits of the distributed Ω-Ants

*The author would like to thank EPSRC for its support by grant GR/M99644.
†The author's work was supported by the 'Studienstiftung des deutschen Volkes'.

command suggestion mechanism are the increased robustness (errors in the distributed computations do not harm the overall mechanism), its resource- and user-adaptability, and its run-time extendibility and modifiability [5].

In this paper we present how we achieve the automation of Ω-ANTS. On the one hand the concurrency enables the integration of external reasoning systems into the suggestion process. External reasoners can either be used to suggest whole subproofs or to compute particular arguments of commands. On the other hand Ω-ANTS can be automated directly by executing suggested commands automatically instead of just presenting them to the user. Thereby it is important to restrict the set of involved commands to those suitable for automation and to fix a certain clock speed determining the period of time the suggestion mechanism may maximally consume for its computations in-between the automated command executions. In any proof state where Ω-ANTS cannot find any new applicable commands it simply backtracks by retracting the lastly executed command.

The Ω-ANTS suggestion mechanism and the Ω-ANTS theorem prover have been developed and implemented within the ΩMEGA theorem proving environment [17]. However, the approach is not restricted to a particular logic, calculus, or theorem proving environment. It can be rather seen as an approach parameterised over the particular calculus it is working for. In this respect the question arises how the designer of the Ω-ANTS agents which have to be provided for each calculus rule can ensure that the modeling guarantees a complete proof search in Ω-ANTS. This question is discussed in the second half of the paper by informally defining properties of agent societies in Ω-ANTS which are necessary to ensure completeness and by giving some examples how these properties are checked in practice.

This paper is organised as follows: Sec. 2 sketches the Ω-ANTS command suggestion mechanism (for further details see [4, 5]), illustrates its declarative agent specification language, and sketches a formal semantics. Sec. 3 describes how external reasoners can be integrated at different layers. In Sec. 4 the Ω-ANTS theorem prover built on top of the suggestion mechanism is introduced and completeness aspects are discussed in Sec. 5. We conclude with discussing some related work and hinting at future work.

2 The Ω-ANTS suggestion mechanism

In this section we sketch the hierarchical, agent-based suggestion mechanism underlying the Ω-ANTS theorem prover. We also discuss the declarative agent specification language supporting the specification and modification of agents at run-time, and sketch how this language can be linked to a formal semantics.

Agent-based architecture The suggestion mechanism originally aims at supporting a user in interactive tactical theorem proving to choose an appropriate proof rule from the generally large set of available ones. It computes and proposes commands that invoke proof rules that are applicable in a given proof state.[1] This is basically done in two steps: firstly, by computing whether there are any possible instantiations for single arguments of a command in the current proof state; and secondly, by gathering those commands for which at least some arguments could be instantiated and presenting them in some heuristically ordered fashion to the user.

An important notion for the Ω-ANTS mechanism is that of a *Partial Argument Instantiations (PAI)* for a command. Considering a command and its corresponding proof rule there is usually a strong connection between the formal arguments of both, i.e. the formal arguments of the command are generally a subset of the formal arguments of the proof rule. As an example we observe the proof rule ∧I and its corresponding command `AndI`:

$$\frac{A \quad B}{A \wedge B} \wedge\mathrm{I} \longrightarrow \frac{\texttt{LConj} \quad \texttt{RConj}}{\texttt{Conj}} \texttt{AndI}$$

Here the command's formal argument `Conj` needs to be instantiated with an open proof node containing a conjunction, `LConj` and `RConj` with nodes containing the appropriate left and right conjuncts, respectively. In general, a command's formal arguments need to be partially instantiated only, in order to be applicable. For instance, `AndI` is also applicable if only the `Conj` argument is provided, resulting in the introduction of two new open proof nodes containing the two conjuncts. Or additionally one of `LConj` or `RConj` or even both could be provided, resulting in the introduction of only one open node or in simply closing the given open conjunction. Thus, we can denote partial argument instantiations for a command as a set relating some of a command's formal argument to actual arguments for its execution. One possible PAI for `AndI` would be $(\texttt{Conj} : x, \texttt{LConj} : y)$ where x and y are proof nodes that contain the appropriate formulas. PAIs can also be seen as functions, indexed by the different command names, with the set of argument-names as domain and the infinite set of possible proof lines and parameters as codomain. For instance, PAIs for `AndI` can be represented as particular functions

$$\mathrm{PAI}^{\texttt{AndI}} : \{\texttt{Conj, LConj, RConj}\} \longrightarrow \mathrm{Prooflines} \cup \mathrm{Parameters} \cup \{\epsilon\}$$

[1] For the remainder of the paper, if we talk about applicability of a command we always mean the applicability to the corresponding proof rule (e.g., a calculus rule, a tactic, a proof planning method, or an external system call) in the given proof state.

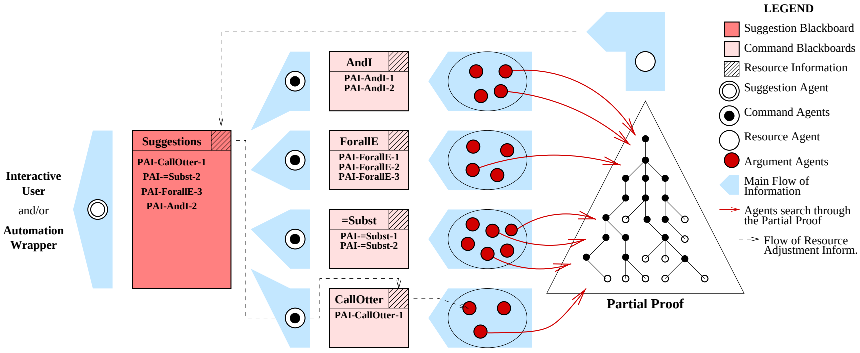

Figure 1. The hierarchical, agent-based Ω-ANTS-architecture.

where ϵ is a special symbol denoting the empty proofline. In these sense the PAI $(\texttt{Conj}:x,\texttt{LConj}:y)$ for `AndI` is realised by a respective function such that $\mathrm{PAI}^{\texttt{AndI}}(\texttt{Conj})=x$, $\mathrm{PAI}^{\texttt{AndI}}(\texttt{LConj})=y$, and $\mathrm{PAI}^{\texttt{AndI}}(\texttt{RConj})=\epsilon$.

The idea of the suggestion mechanism is to compute in each proof state for each command PAIs as complete as possible, to determine which commands are applicable, and then to give preference to those, e.g., with the most complete PAIs. The first task is done by societies of *Argument Agents* (rightmost circles in Fig. 1) where one society is associated with each of the commands. Each argument agent is associated with one or several of the command's formal arguments and has a specification for possible instantiations of these arguments. Its task is to search for proof nodes in the partial proof or to compute parameters according to its specification. Argument agents exchange results via *Command Blackboards* (for each command one command blackboard is provided) using PAIs as messages. Every argument agent commences its computations only when it finds a PAI on the command blackboard that contains instantiations of arguments that are relevant for its own computations.

For example, the `AndI` argument agent associated with Conj searches the partial proof for an open node containing a conjunction and, once it has found one, say in node x, it places a respective PAI $(\texttt{Conj}:x)$ on the command blackboard. Now the agents for `LConj` and `RConj` can use this result in order to look simultaneously for nodes in the given partial proof containing the appropriate left or right conjunct. Each argument agent only reads old suggestions and possibly adds expanded new suggestions, thus there is no need for conflict resolutions between the agents.

On top of the layer of argument agents are the *Command Agents* (dotted circles). Their task is to monitor the command blackboard associated with the command and to heuristically order the PAIs from most promising (e.g., most complete) to least promising. Whenever their heuristics indicate that there is a new best PAI on the command blackboard they pass it to the *Suggestion Blackboard*. The suggestion blackboard itself is again monitored by the *Suggestion Agent* (leftmost double circle) which sorts the entries with respect to its heuristic criteria and presents them to the user.

When the Ω-ANTS mechanism is started all command blackboards are initialised with the empty PAI. The agents then autonomously search for applicable commands and the newest suggestions are successively presented to the user. At any point a command can be executed and when the proof state has actually been changed the Ω-ANTS mechanism is re-initialised in order to compute new suggestions for the modified proof state. Ω-ANTS can also be used to respond to particular user queries, i.e. the user can interactively specify certain argument instantiations and the mechanism tries to complete these.

The whole mechanism can be adjusted during run-time by changing sorting heuristics for the command blackboards and the suggestion blackboard or by removing, adding, or modifying argument agents. Moreover, Ω-ANTS employs a resource mechanism that automatically disables and enables argument agents with respect to their usefulness and performance in particular proof states. Although not depicted here, the mechanism also contains classification agents whose purpose is to classify the focused subproblem in terms of logic and mathematical theory is belongs to. This information is communicated within the blackboard architecture enabling agents to decide whether they are appropriate (i.e. should be active) in the current proof state or not. See [5] for further details.

A Declarative Agent Specification Language In Ω-ANTS only the argument agents need to be explicitly specified. All other agents are then generated automatically (certain heuristics may be adapted by the user, though). Argument agents are implemented with a Lisp-like declarative language such as the following two argument agents for the `AndI` command:

$\mathfrak{A}_1$:
```
(agent~defagent AndI c-predicate
   (for Conj) (uses )
   (exclude LConj RConj)
   (definition
     (logic~conjunction-p Conj)))
```

$\mathfrak{A}_4$:
```
(agent~defagent AndI s-predicate
   (for RConj) (uses Conj)
   (definition
     (logic~right-conjunct-p RConj Conj))))
```

The agent $\mathfrak{A}_1$ is defined as a `c-predicate` agent, indicating that it will always restrict its search to open proof nodes, i.e., possible conclusions. `s-predicate` agents like $\mathfrak{A}_4$ in contrast search the support nodes for possible premises. The proof nodes $\mathfrak{A}_1$ is looking for are instantiations of the argument `Conj`, given in the `for`-slot. The empty `uses`-slot indicates that

$$
\begin{array}{llcl}
\mathfrak{A}_1: & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\},\{\mathrm{LConj},\mathrm{RConj}\}} & := & \lambda Conj_{\blacksquare}(Conj \equiv A \wedge B) \\
\mathfrak{A}_2: & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\mathrm{LConj}\},\{\mathrm{RConj}\}} & := & \lambda Conj_{\blacksquare}(Conj \equiv A \wedge B)\ \&\ (\mathrm{LConj} \equiv A) \\
\mathfrak{A}_3: & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\mathrm{RConj}\},\{\mathrm{LConj}\}} & := & \lambda Conj_{\blacksquare}(Conj \equiv A \wedge B)\ \&\ (\mathrm{RConj} \equiv B) \\
\mathfrak{A}_4: & \mathfrak{S}^{\{\mathrm{RConj}\}}_{\{\mathrm{Conj}\},\{\}} & := & \lambda RConj_{\blacksquare}(\mathrm{Conj} \equiv A \wedge B)\ \&\ (RConj \equiv B) \\
\mathfrak{A}_5: & \mathfrak{S}^{\{\mathrm{LConj}\}}_{\{\mathrm{Conj}\},\{\}} & := & \lambda LConj_{\blacksquare}(\mathrm{Conj} \equiv A \wedge B)\ \&\ (LConj \equiv A) \\
\mathfrak{A}_6: & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\mathrm{LConj},\mathrm{RConj}\},\{\}} & := & \lambda Conj_{\blacksquare}(Conj \equiv A \wedge B)\ \&\ (\mathrm{LConj} \equiv A)\ \&\ (\mathrm{RConj} \equiv B)
\end{array}
$$

Figure 2. A society of argument agents for command `AndI`.

the agent does not require any already given argument suggestions in a PAI for its computations. The `exclude`-slot on the other hand determines that this agent must not complete any PAI that already contains an instantiation for arguments `LConj` or `RConj`. In the special case of $\mathfrak{A}_1$ this means the agent is exactly triggered by the empty PAI. The idea for this exclusion constraint is to suppress redundant or even false computations.

The full set of argument agents for the `AndI` command is given in Fig. 2 in a specification meta-language. `c-predicate` and `s-predicate` agents are denoted by $\mathfrak{C}$ and $\mathfrak{S}$ respectively, the superscript set corresponds to the `for`-list, and the `uses`- and `exclude`-list to the first and second index. The subset of the nodes in a partial proof that will be detected by each argument agent can be formally described by a λ-term (characteristic function). When running over the partial proof the agents use these characteristic functions to test each node before possibly returning an expanded PAI. A and B are free meta-variables. $\equiv$ and & are symbols of the meta-language with the meaning, for instance in agent $\mathfrak{A}_2$, that given an arbitrary formula A instantiating argument LConj then $Conj$ has to be of form $A \wedge B$, i.e. the left hand side of $Conj$ is determined by the already given suggestion LConj whereas is right hand side is still *free*.

This attempt at a formal semantics for our agent definitions by assigning characteristic functions to them does not yet address the agents functional behaviour (they pick up & return potentially modified PAIs) nor does it formally regard the uses and exclude-restrictions. This is the idea of the λ-expression for agent $\mathfrak{A}_2$ below. Assuming that PAIs are represented as functions this term denotes that $\mathfrak{A}_2$ picks up certain PAIs on the blackboard and returns possibly modified ones while using an (extended/modified) characteristic function in the previous sense as filter. Here the [.]-brackets denote a function which accesses the formula content of the proofline given as an argument to it (note that PAIs map argument names to prooflines, while here we want to talk about the formulas of the prooflines[2]).

[2]In other parts of this paper we do not take this so serious and assume that the user

$\lambda\ PAI\ \blacksquare\ \lambda\ Conj_{\text{open}}\ \blacksquare$

$$
\begin{array}{llll}
\text{if } PAI(\text{Conj}) \equiv \epsilon\ \&\ PAI(\text{LConj}) \not\equiv \epsilon\ \&\ PAI(\text{RConj}) \equiv \epsilon & & & \\
\text{then} & \text{if } [Conj] \equiv\ \text{A} \wedge \text{B}\ \&\ [PAI(\ \text{LConj})] \equiv \text{A} & & \\
 & \text{then } PAI|_{\{\text{LConj, RConj}\}} \cup \{\text{Conj} \mapsto Conj\} & & \rightarrow \text{new ext. PAI} \\
 & \text{else } PAI & & \rightarrow \text{no new PAI} \\
 & \text{fi} & & \\
\text{else } PAI & & & \rightarrow \text{no new PAI} \\
\text{fi} & & &
\end{array}
$$

3 Integration of External Reasoning Systems

The following four examples illustrate how external reasoners can be integrated into Ω-ANTS. The first row presents four inference rules and the second the corresponding commands which we want to model in Ω-ANTS.

$$
\frac{P_1 \quad \ldots \quad P_n}{C}\ \begin{matrix}\mathit{Otter}\\ \mathit{Mace}\end{matrix} \qquad \frac{A \quad B \Rightarrow C}{C}\ \begin{matrix}\mathit{mp\text{-}mod\text{-}Otter}(A \Rightarrow B)\\ \mathit{mp\text{-}mod\text{-}CAS}(A \overset{\text{simpl}}{\rightarrow} B)\end{matrix}
$$

$$
\frac{\texttt{Prem}_1 \quad \ldots \quad \texttt{Prem}_n}{\texttt{Conc}}\ \begin{matrix}\texttt{Otter}\\ \texttt{Mace}\end{matrix} \qquad \frac{\texttt{Left} \quad \texttt{Impl}}{\texttt{Conc}}\ \begin{matrix}\texttt{mp-mod-Otter(Impl-Prob)}\\ \texttt{mp-mod-CAS(Simpl-Prob)}\end{matrix}
$$

The first two rules describe the integration of the first-order theorem prover OTTER and the propositional logic decision procedure MACE. These commands may be used in a given proof state in order to justify a goal from its premises by the application of one of these external systems. The next two rules describe a situation where external reasoners are used within an inference modulo, in our particular case modus ponens modulo the validity of an implication (to be checked by OTTER) and modus ponens modulo the simplifiability of a proposition (to be analyzed by a computer algebra system). For instance, sensible instances of these commands in a concrete proof situation would be: `Left` $\leftarrow \forall x.p(x) \wedge q(x)$, `Impl` $\leftarrow p(a) \Rightarrow r(a)$, `Conc` $\leftarrow r(a)$, and `Impl-Prob` $\leftarrow (\forall x.p(x) \wedge q(x)) \Rightarrow p(a)$ for `mp-mod-Otter`, and `Left` $\leftarrow$ continuous$(\lambda x.1 - \cos^2(x))$, `Impl` $\leftarrow$ continuous$(\lambda x.\sin^2(x)) \Rightarrow$ something$(\lambda x.\sin^2(x))$, `Conc` $\leftarrow$ something$(\lambda x.\sin^2(x))$, `Simpl-Prob`$\leftarrow$continuous $(\lambda x.1 - \cos^2(x)) \overset{\text{simpl}}{\rightarrow}$ continuous$(\lambda x.\sin^2(x))$ for `mp-mod-CAS`. The idea is that the external systems are used to check the 'modulo'-side-conditions of these rules. Note, that in contrast to the theorem proving modulo approach described in [9] we explicitly facilitate and support the integration of non-decision procedures; a strict separation of deduction and computation is not needed due to the distribution and resource-guidance aspect of the Ω-ANTS mechanism.

We are here not concerned with correctness issues for the integration of the external systems. However, since we are working in the ΩMEGA

recognises whether we address a proof line or its formula content from the context.

environment we can make use of the work already done in this area that ensures the correctness by translating proofs or computations from external reasoners into primitive inference steps of ΩMEGA [16, 19].

If we, for example, consider the `Otter` and the `mp-mod-CAS` command we can observe two different ways of integrating the external reasoners into agents: For the `Otter` command one agent attacks the focused open goal in-between user interactions and as soon as OTTER finds a proof the application of this command is suggested to the user. Thus, the agent employs the external system to prove an open sub-problem. Similarly, other external reasoners can be integrated.

In case of the `mp-mod-CAS` command an agent will first look for appropriate implication proof nodes with respect to the open goal. This agent's results trigger another agent that which employs an integrated computer algebra system to look for appropriate proof nodes as instances for argument `Left`. More precisely, the latter agent checks whether a proof node can be matched with the antecedent of the implication with respect to algebraic simplification of sub-terms. Hence, the agent uses an external reasoner only to find possible instantiations of arguments.

4 Automation

The Ω-ANTS suggestion mechanism of Sec. 2 can be automated into a full-fledged proof search procedure by embedding the execution of suggested commands into a backtracking wrapper. The algorithm is given in Fig. 3.

The basic automation performing a depth first search is straight forward: The suggestion mechanism waits until all agents have performed all possible computations and no further suggestions will be produced and then executes the heuristically preferred suggestion (1a&2). When a proof step is executed and the proof is not yet finished, the remaining suggestions on each command blackboard are pushed on the backtracking stack (3). In case no best suggestion could be computed Ω-ANTS backtracks by popping the first element of the backtrack stack and re-instantiating its values on the blackboards (6). The proof is constructed as an explicit proof plan data structure of ΩMEGA [8]. It enables to store proofs in a generalised natural deduction format, i.e. proof steps cannot only be justified by basic natural deduction rules but by abstract tactics or computations of external reasoners as well. Moreover, the proof plan data structure supports the expansion of externally computed proofs into primitive inference steps and thus the check for correctness as well as the storage of information for the automation loop directly in the proof object.

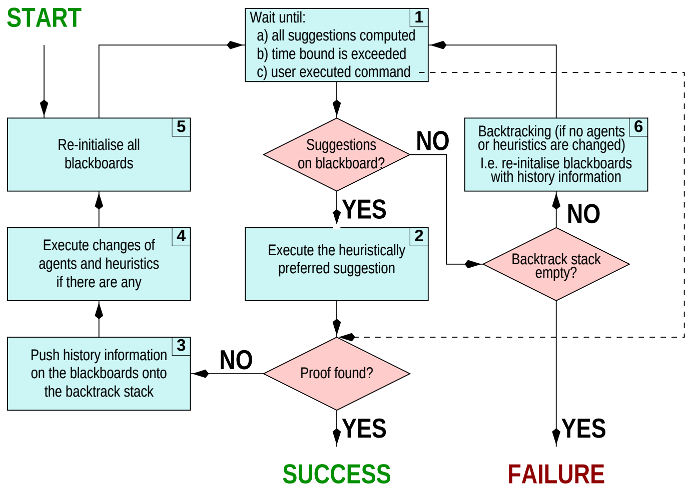

Figure 3. Main-loop of the Ω-ANTS theorem prover.

The simple automation loop is complicated by the distinct features of Ω-ANTS: (i) some agents can perform infinite or very costly computations, (ii) commands can be executed by the user parallel to the automation, and (iii) the components of Ω-ANTS can be changed at run-time. Furthermore, the automation can be suspended and revoked especially in order to perform the latter two interaction possibilities in a coordinated way.

We avoid that Ω-ANTS is paralysed by agents that get stuck in infinite computations by giving a time limit after which the best command, suggested so far, is executed (cf. step 1b). However, such a proof step is treated special when backtracking, since then the blackboards will be re-instantiated with all the values of the proof step, i.e. containing the executed command as well. This way there is a second chance for agents that could not contribute the first time to add information. The question how the Ω-ANTS theorem prover can avoid to get lost on an infinite branch in the search space without ever backtracking will be addressed in the completeness discussion in Sec. 5.

If a command has been executed by the user the loop proceeds immediately with saving the blackboards' history without executing another command (1c). When backtracking the whole history on the last step is reinstantiated onto the blackboards, possibly containing also the command executed by the user, in order not to loose possible proofs (1c&6).

One main feature of Ω-ANTS is its run-time adaptability by adding or deleting agents or changing the filter and sorting heuristics used by the suggestion and commands agents. These changes also take effect when running the automation wrapper (4). The automation wrapper can be suspended by the user at any time, for instance, in order to analyzed the current proof state and to add, change or remove certain agents from the suggestion mechanism. It can then be resumed using all the information computed so far.

We briefly summarise the user interaction facilities inherited by the Ω-ANTS prover from the Ω-ANTS suggestion mechanism:

Pure user interaction/mixed initiative reasoning: In automation mode the entries on the suggestion blackboard are (theoretically[3]) steadily visible to the user, who can interfere with the automation wrapper by executing a command before the automation wrapper does.

Adjustment of resource bounds: The user may want to actively modify the resource bounds (time, memory, deactivation threshold) in order to adapt the system to particular needs.

Disable/resume single agents: Ω-ANTS allows to disable/resume single agents, agent societies, or the whole mechanism at run-time.

Modification/addition of argument agents: The user may want to specify and load new agents at run-time or modify the definition of already given agents. This is supported by the declarative agent-specification language.

Modification of command/suggestion agents: In order to influence the provers search through the search space the user may want to choose different heuristics and sorting criteria for these agents.

The Ω-ANTS system has been applied to automate the propositional logic fragment of the normal form natural deduction calculus NIC [7], see [2] for more details. We currently experiment with the full first-order fragment of NIC. The integration of external reasoners has been tested with the propositional logic prover MACE, the first-order provers OTTER and SPASS, the higher-order prover TPS, and the computer algebra system MAPLE. The theorems we are working with are still all relatively simple and nothing any of the involved systems is not able to solve on its own. The computations involved are mainly to solve equations and compute derivatives.

5 Ω-ANTS and Completeness

In this section we introduce and discuss some notions that are necessary to characterise and guarantee completeness and soundness of a theorem prover based on Ω-ANTS with respect to the underlying calculus. The discussion

[3]In our experiments with the NIC calculus, the theorem prover is unfortunately much faster than the graphical user interface to allow a synchronised displaying.

is rather informal since we have yet to define completely formal syntax and semantics for our agent specification language. However, the following shall both give an intuition for the properties that need to be considered and contributes to a better understanding of Ω-Ants.

Given a theoretically complete calculus, how can it be modeled in Ω-Ants such that completeness is still assured in the mechanism? Note, that we do not address the theoretical completeness of the underlying calculus itself, in fact we do not even need to specify here what particular logic and calculus we are interested in. We rather aim to ensure that each calculus rule application that is theoretically possible in a given proof state can indeed be determined and suggested by the Ω-Ants mechanism. In particular we will discuss two different notions of completeness in this sense, namely *interaction completeness* and *automation completeness*. This is due to twofold bias of the Ω-Ants system as a suggestion mechanism and as an automated theorem prover. The authors admit that naming these properties also 'completeness' might be slightly misleading. However, automation (interaction) completeness of the agent societies involved taken together with the 'theoretical (logical) completeness' of a calculus implies that a complete proof search is actually supported by Ω-Ants.

Theoretical completeness investigations typically assume non-limited resources like computation time and space. In our case the resources available to the Ω-Ants-system in-between the command executions are crucial wrt. completeness as well. However, for the time being we neglect points possibly interfering with this assumption, in particular cases 1(b) or 1(c) of the prover's main-loop in Fig. 3 and the existence of agents with calls to undecidable procedures such as the Otter agent in Sec. 3.

Automation Completeness Automation completeness depends in the first place on the *suggestion completeness* of the argument agent societies associated with each rule: A society of suggestion agents working for a single command C is called *suggestion complete* wrt. a given calculus, if in any possible proof state all PAIs of a command necessary to ensure completeness of the calculus can be computed by the mechanism. Under the resource abstraction assumption from above suggestion completeness requires that each particular agent society consists of *sufficiently* many individual suggestion agents and that their particular definitions are *adequate* wrt. the structural dependencies and side-conditions of the respective calculus rule. *Adequacy* basically excludes wrong agent specifications, while *Sufficiency* refers to the ability of an agent society to cooperatively compute each applicable PAI in a given proof state.

We call a command agent *non-excluding* if it indeed always reports at least one selected entry from the associated command blackboard to the suggestion blackboard as soon as the former contains some applicable PAIs. And the suggestion agent is non-excluding if it always reports the complete set of entries on the command blackboard to the automation wrapper. This ensures that computed PAIs are actually propagated in the mechanism.

We additionally have to ensure that the proof search is organised in a *fair* way by ensuring that the execution of an applicable PAI suggested within a particular proof step cannot be infinitely long delayed. The fairness problem of Ω-ANTS is exactly the same as in other theorem proving approaches performing depth first search. In our experiments with the propositional logic fragment of the NIC this problem did not occur as the considered fragment defines a decision procedure. However to ensure that the prover does not get lost on infinite search path when working with the full first-order fragment of NIC we chose iterative deepening search.

Our mechanism can then be called automation complete wrt. to a given calculus C if (i) the agent societies specified are suggestion complete wrt. C, and (ii) the command agents for C and the suggestion agent are non-excluding, (iii) the search procedure is fair and (iv) the resource bounds and deactivation threshold are chosen sufficiently high, such that each agents computation terminates within these bounds.

We illustrate the notions of adequacy and sufficency in more detail with the example of the `AndI` agents. We claim that the agents $\mathfrak{A}_1 \ldots \mathfrak{A}_6$ of Fig. 2 are both (a) adequate and (b) sufficient to apply `AndI` (whenever possible) in automated proof search.

(a) To show that all computable suggestions are indeed applicable we check that each agent produces an adequate predicate if all arguments of the uses slot are instantiated correctly. We observe this in the case, of agent $\mathfrak{A}_2$ when applying it to a PAI of the form (`LConj`:a). Here a is an arbitrary but fixed term. The resulting predicate is $Conj{\equiv}a{\wedge}B$ which permits all conjunctions with left conjunct a and is therefore adequate.

After checking adequacy of all single agents we have to ensure adequacy of cooperation between agents. That is, to show that no incorrect PAIs can be assembled by cooperation of agents with correct predicates. Here we are only concerned with agents whose `for`-, `uses`-, and `exclude`-list does not contain all possible arguments of the command, thus in our case agents $\mathfrak{A}_4$ and $\mathfrak{A}_5$. It can be easily seen that even if, for instance, $\mathfrak{A}_4$ is applied to a PAI already containing an instantiation for `LConj`, adding an appropriate instantiations for `RConj` will maintain the PAI's applicability, provided it was correct to begin with.

(b) To ensure sufficiency we have to show that each PAI of `AndI` necessary for automation can (cooperatively) be computed. In automatic mode the NIC calculus is intended for pure backward search and thus the possible PAIs are of the form[4] i) (Conj: $a \wedge b$), ii) (Conj: $a \wedge b$, LConj: a), iii) (Conj: $a \wedge b$, RConj: b), or iv) (Conj: $a \wedge b$, LConj: a, RConj: b), where a and b are arbitrary but fixed formulas occurring in a partial proof P. We representatively discuss case ii) and verify that each PAI of form S = (Conj: $a \wedge b$, LConj: a) that is applicable in P will actually be computed. As S is applicable, P must contain an open node containing $a \wedge b$ together with a support node containing a. Initially the command blackboard contains the empty PAI () to which only $\mathfrak{A}_1$ can be applied. Provided the underlying implementation, i.e. the function `logic~conjunction-p` is correct, $\mathfrak{A}_1$'s predicate suffices to compute (Conj: $a \wedge b$). This PAI in turn triggers the computations of $\mathfrak{A}_4$ and $\mathfrak{A}_5$ with the respective instantiated predicates $RConj{\equiv}b$ and $LConj{\equiv}a$. Since the latter is true on the support node containing a, $\mathfrak{A}_5$ returns the PAI in question.

When checking all other cases we can observe that for the automation mode (where pure backward reasoning is assumed) the agents $\mathfrak{A}_1$,$\mathfrak{A}_4$, and $\mathfrak{A}_5$ are already sufficient. And indeed the other three agents are needed to support user interaction, only. For instance, the user can apply Ω-ANTS to complete a particular PAI like (LConj: a) which will trigger the computations of agent $\mathfrak{A}_2$.

Interaction Completeness Interaction completeness of a calculus implies that one never has to rely on another interaction mechanism besides Ω-ANTS in order to perform possible proof steps within a given calculus. Therefore, we have to show that all possible PAIs to apply a rule interactively can be computed. This is generally a stronger requirement than for automation completeness as can be easily observed with our `AndI` example. When automated the NIC calculus strictly performs backward search and only the PAIs (i)—(iv) given above are legitimate. However, when using the calculus interactively forward reasoning (i.e. a PAI of the form (LConj: a, RConj: b)) is a perfectly legal option. But it can be easily seen that this PAI cannot be computed with the given agent society and thus $\{\mathfrak{A}_1 \ldots \mathfrak{A}_6\}$ are not interaction complete.

When dealing with interaction completeness we have also to consider all possible initialisations of the command blackboards. While in automation mode the blackboards are always initialised with the empty PAI, the user can ask Ω-ANTS interactively to complete a particular PAI (such as (LConj: a)) which is then used as initial value on the blackboard. It is necessary to show sufficiency and adequacy for all possible initialisations.

Soundness Should not the *soundness* aspect be addressed here as well? Our answer is no, as we presuppose that the underlying theorem proving

[4] PAIs are essentially sets and thus the order of the particular entries is not important.

environment takes care of a sound application of its own proof rules. Furthermore, in systems such as ΩMEGA soundness is always only guaranteed on the level of primitive inferences and not necessarily for all proof methods etc. involved. Thus, soundness requirements when computing suggestions for methods that do not necessarily lead to a correct proof would not make sense. Thus, instead of *logical soundness* we are rather interested in the notion of *applicability*. This notion relates the PAIs computed by Ω-ANTS to the particular side-conditions of the underlying proof rules (whether they are logically sound or not).

The effect of non-applicable PAIs suggested to the user or the automation wrapper might lead to failure when applying the respective command. In the current implementation such a failure will simply be ignored and the responsible PAI is discarded. However, too many non-executable suggestions might negatively influence the mechanisms user-acceptance and especially the performance of the automation wrapper.

6 Related Work

There exist several theorem proving environments where a mixture of interactive and partial automated proving is supported. In systems such as PVS [18] and HOL [13] special tactics are available that can be used to automatically solve certain problems. These tactics are essentially proof procedures build on top of the primitive inferences of the respective systems but do not directly construct a proof in terms of primitive inferences, although the automated parts can be, at least in the case of HOL, expanded. Moreover, there is no possibility for a user of the system to change the behaviour of the automation tactic during its application. In the TPS system [1] interaction and automation can also be interleaved and any automatic proving attempt can be interrupted, its behaviour changed and restarted by the user. The automation is achieved by using a mating search technique that is substantially different from the natural deduction calculus that is used for interactive proving. Finally, an approach to achieve automation in an interactive environment is to enable the use of external reasoners which is, for instance, one of the features of the ΩMEGA system [17]. However, without the Ω-ANTS part, application of rules, tactics and external reasoners cannot be automated.

As an environment that is especially designed to support the combination of interactive and automated theorem proving together with the use of already existing reasoning systems, is the Open Mechanised Reasoning System [12, 11] that has been extended to facilitate computer algebra systems [6]. While the concept of a reasoning structure to represent explicit proof states is similar to our concept of a proof object, external reasoners are connected as *plug-and-play* components which requires significant changes to their control components and therefore complicates the use of existing technology.

7 Conclusion

We presented the Ω-ANTS theorem prover build on top of the agent-based Ω-ANTS suggestion mechanism. This theorem prover inherits interesting features from the underlying suggestion mechanism and due to the distribution of computations down to a very fine-grained layer (e.g. reasoning about potential instances of single arguments of the considered inference rules) it especially supports the integration of external reasoning systems at various layers. We have illustrated that the Ω-ANTS architecture especially supports deduction modulo computation/deduction performed by external reasoners. As the same suggestion mechanism that supports user-interaction is now also used as the main part of the automated theorem prover's inference machine the architecture also supports a close integration of interactive and automated theorem proving. This is underlined by the various interaction facilities the Ω-ANTS prover already supports. The system can be seen as an open approach that is parameterised over the particular calculus it is working for (and note that it is only in a technical sense restricted to the ΩMEGA environment in which it has been developed). The calculus it is working for can even be modified/extended at run-time, making our system in the long-run also interesting for the integration of components aiming at learning new inference rules from past proof experience [15]. The learned rules could then be dynamically added to the running system.

Immediate further work is a more rigorous formalisation of the agent specification language as well as to formally model the connection between Ω-ANTS and underlying calculi. Other future work is to analyse whether our system could benefit from a dynamic agent grouping approach as described in [10] and whether it can fruitfully support the integration of proof critics as discussed in [14]. The Ω-ANTS system is also employed as the basis of the resource-guided and agent-based proof planning approach [3], currently under development. Extending the Ω-ANTS system this approach also focuses on the cooperation aspect between integrated external reasoners and addresses the question how an agent-based proof planner can be sensibly guided by a resource mechanism.

Acknowledgement The authors thank John Byrnes for his support in realizing the NIC calculus in Ω-ANTS. We furthermore thank S. Autexier, M. Kerber, M. Jamnik, and M. Hübner for fruitful discussions.

References

[1] 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.

[2] C. Benzmüller, J. Byrnes, and V. Sorge. Ω-ANTS for interactive ATP. Unpublished draft: `www.ags.uni-sb.de/~chris/oants-nic00.ps.gz`.

[3] C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge Towards Concurrent Resource Managed Deduction. Cognitive Science Research Paper CSRP-99-17, University of Birmingham, 1999.

[4] C. Benzmüller and V. Sorge. A Blackboard Architecture for Guiding Interactive Proofs. *Proc. of AIMSA'98*, LNAI 1480, Springer, 1998.

[5] C. Benzmüller and V. Sorge. Critical Agents Supporting Interactive Theorem Proving. *Proc. of EPIA-99*, LNAI 1695, Springer, 1999.

[6] P. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. *Fundamenta Informaticae*, 39(1–2), 1999.

[7] J. Byrnes. *Proof Search and Normal Forms in Natural Deduction.* PhD thesis, Dep. of Philosophy, CMU, Pittsburgh, PA, USA, 1999.

[8] L. Cheikhrouhou and V. Sorge. $\mathcal{PDS}$ — A Three-Dimensional Data Structure for Proof Plans. In *Proc. of ACIDCA'2000*, Monastir, Tunisia, 2000.

[9] G. Dowek, Th. Hardin, and C. Kirchner. Theorem proving modulo. Rapport de Recherche 3400, INRIA, France, 1998.

[10] M. Fisher and M. Wooldridge A Logical Approach to the Representation of Societies of Agents. In N. Gilbert and R. Conte, editors, Artificial Societies. UCL Press, 1995.

[11] F. Giunchiglia, P. Bertoli, and A. Coglio. The OMRS project: State of the Art. *Electronic Notes in Theoretical Computer Science*, 15, 1998.

[12] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories - Towards an Architecture for Open Mechanized Reasoning Systems. In *Proc. of Frontiers of Combining Systems*, pages 157–174, 1996.

[13] M. J. C. Gordon and T. F. Melham. *Introduction to HOL.* Cambridge University Press, Cambridge, United Kingdom, 1993.

[14] A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. Journal of Automated Reasoning, 16(1–2):79–111, 1996.

[15] M. Jamnik, M. Kerber, and C. Benzmüller. Towards Learning new Proof Methods in Proof Planning. In this volume.

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

[17] The ΩMEGA-group. ΩMega: Towards a Mathematical Assistant. *Proc. of CADE–14*, LNAI 1249, Springer, 1997.

[18] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. In *Computer-Aided Verification, CAV '96*, LNCS 1102, pages 411–414. Springer, 1996.

[19] V. Sorge. Non-Trivial Computations in Proof Planning. In *Proc. of Frontiers of Combining Systems*, LNCS 1794. Springer, 2000.