# System Description: Ω-ANTS for interactive ATP

Christoph Benzmüller[1]*, John Byrnes[2], and Volker Sorge[3]

[1] School of Computer Science, The University of Birmingham
Edgbaston, Birmingham B15 2TT, England
[2] Department of Philosophy, Carnegie Mellon University
Pittsburgh, PA 15213, USA
[3] Fachbereich Informatik, Universität des Saarlandes,
D-66041 Saarbrücken, Germany

## 1 Introduction

This paper describes Ω-ANTS[1], an approach for integrating interactive and automated proof search. Our system which has been prototypically realized within the ΩMEGA environment [4] extends the concurrent, agent based suggestion mechanism for interactive theorem proving described in [1,2] by a backtracking wrapper that turns it into an automated theorem prover with interesting features like anytime character and run-time extensibility. The side conditions and the control knowledge wrt. the particularly implemented calculus are completely encoded in the specifications of the numerous concurrent (software) agents, which individually perform quite trivial deduction/suggestion tasks.

In order to demonstrate that even nontrivial proof calculi can be realized in Ω-ANTS and especially in order to employ a human oriented approach with interesting proof automation capabilities we have implemented the intercalation calculus NIC [3] for efficient natural deduction proof search.

## 2 The Ω-ANTS system

We outline the architecture of Ω-ANTS by first sketching the agent-architecture of the underlying suggestion mechanism and then describing how this mechanism can be automated into a full-fledged proof search procedure. For a more detailed introduction to the pure suggestion mechanism see [1,2].

*Architecture* The suggestion mechanism originally aims at supporting a user in interactive theorem proving by computing and proposing possibly applicable commands. The suggestions are thereby computed by two layers of societies of autonomous concurrent agents which exchange relevant results via blackboards.

In an interactive theorem prover such as ΩMEGA one has generally a command associated with each proof tactic to invoke it. Commands are applied to a set of arguments such as proof lines or terms that specify how the respective tactic is to be employed in the current proof state. Depending on the particular tactic the arguments of the command have then to be of a certain form. For example, the command invoking the ND rule ∧I, i.e. the splitting of a conjunctive

* The author would like to thank EPSRC for its support by grant GR/M99644.

[1] ΩMEGA's Agent-based Natural deduction Theorem proving System

goal, can only be applied to a proof line containing a conjunction and, if available in the current proof state, to lines containing the right and left conjunct. We can use these structural dependencies in order to find and suggest possible arguments for each command with respect to the given proof state. Having those suggestions available one has then also knowledge which commands are applicable altogether.

The suggestion process is carried out by a two layered architecture of autonomous agents. On the bottom layer we have associated with each command a society of *Argument Agents*. Each argument agent is appointed to one or several arguments of the command and has a declarative specification of their shape and structural dependencies. With the help of this specification and if necessary with information already provided by other agents it can retrieve possible instantiations of the argument from the current proof state either by searching for proof lines or by computing some additional parameter (e.g., a term). The argument agents of each society communicate via a *Suggestion Blackboard*. For example, in the case of the $\wedge$I command one agent searches the open lines of the partial proof for conjunctions and, once it has found some, two other agents can use this result in order to look simultaneously for lines containing the appropriate left or right conjunct. Since each argument agent only reads old suggestions and possibly adds expanded new suggestions, there is no need for conflict resolutions between the agents.

On the second layer of the mechanism is a society of *Command Agents* where every agent is linked to exactly one command. Its task is to initialize (whenever the proof state has changed) and monitor the suggestion blackboard associated with the command. Its intention is to select among the entries of the blackboard the most promising (e.g., most complete) and to pass it, enriched with the corresponding command name to the *Command Blackboard*. Thus, the command blackboard accumulates all commands that can be applied within a given proof state together with appropriate argument instantiations. The command blackboard itself is again monitored by the *Suggestion Agent* which sorts the entries with respect to heuristic criteria and presents them to the user.

*Automation* Automation of this process is achieved in a straight forward way: We always automatically execute the (according to the given heuristics) best command suggestion on the command blackboard. However, since the whole mechanism depends on concurrency, we determine when exactly the next proof step can be performed by: (1) either knowing that all agents performed all possible computations and no further suggestion will be produced or (2) if a certain time bound is exceeded. When a proof step is executed the remaining suggestions on each suggestion blackboard are saved. Hereby the currently applied suggestion (together with all suggestions it subsumes) is either removed or is kept, in case not all argument agents have tried to complete the suggestion so far, i.e. the command was executed due to reaching the time limit. We do backtracking by re-instantiating the suggestion blackboards with the saved suggestions and recommencing computation.

In case not all possible completions for the last applied suggestion were computed they might be supplemented in the second phase. However, if a command has been applied in exactly the same fashion twice, it will not be applied again,

even if not all of its agents have contributed to its completion yet. Thereby we avoid infinite loops in case some agent gets stuck during its computations.

The automation wrapper can be suspended by the user at any time, for instance, in order to analyze 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.

## 3 NIC: Searching for normal natural deductions

The calculus NIC [3] allows one to search directly for normal proofs in the natural deduction calculus of Prawitz [5]. Restriction to normality allows for restricted branching, resulting in more efficient search. In addition, the particular version of the calculus given here employs additional restrictions on the application of disjunction elimination and the classical indirect rule.

A proof line is either of the form $\Gamma \vDash^{\mathrm{e}} A$ (an *e-line*) or of the form $\Gamma \vDash^{\mathrm{i}} A$ (an *i-line*), where $\Gamma$ (the *context*) is a set of formulas and $A$ is a formula. We write $\Gamma \vdash A$ (an unmarked line) to indicate a line which may be either an e-line or an i-line. No line may be both an e-line and an i-line. Negation is defined: $\neg A \equiv A \to \bot$. The rules are as follows:

$$\frac{\Gamma \vDash^{\mathrm{e}} A \wedge B}{\Gamma \vdash A}\ \wedge\mathrm{E}_l \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vDash^{\mathrm{i}} A \wedge B}\ \wedge\mathrm{I} \qquad \frac{\Gamma \vDash^{\mathrm{e}} A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \to\mathrm{E} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vDash^{\mathrm{i}} A \to B}\ \to\mathrm{I}$$

$$\frac{\Gamma \vdash A}{\Gamma \vDash^{\mathrm{i}} A \vee B}\ \vee\mathrm{I}_l \qquad \frac{\Gamma \vDash^{\mathrm{e}} A \vee B \quad \Gamma, A \vDash^{\mathrm{e}} C \quad \Gamma, B \vDash^{\mathrm{e}} C}{\Gamma \vdash C}\ \vee\mathrm{E}_e \qquad \frac{A \vee B\ ?\ A}{\Gamma \vdash A}\ \vee\mathrm{E}_l \qquad \frac{\Gamma, \neg A \vDash^{\mathrm{e}} \bot}{\Gamma \vDash^{\mathrm{i}} A}\ \bot_{\mathbf{c}}$$

There is a symmetric $R_r$ for each rule $R_l$. $\vee\mathrm{E}_e$ is not general enough to allow for complete search. Complete search without excessive branching for disjunction elimination is achieved by including the tactic $\vee$E below, which makes use of occurrences of the dummy rules[2] $\vee\mathrm{E}_l$ and $\vee\mathrm{E}_r$. No rule applies to the premise of $\vee\mathrm{E}_l$ [$\vee\mathrm{E}_r$]; they are used only to accumulate proof context information for $\vee$E.

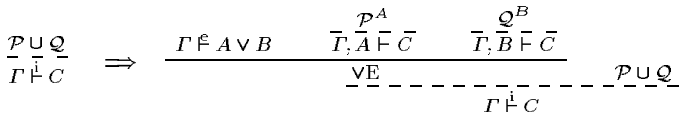
Each rule with a premise of the form $\Gamma \vDash^{\mathrm{e}} C$ has the side condition that $C$ is a strictly positive subformula[3] of some formula in $\Gamma$ (briefly, $C \leq \Gamma$). $\vee\mathrm{E}_l$ [$\vee\mathrm{E}_r$] has the side condition that $A \vee B \leq \Gamma$. $\bot_{\mathbf{c}}$ has the side condition that $A$ is a disjunction or an atomic formula. The search procedure may not repeat a line on a thread. A line of the form $\Gamma, A \vdash A$ is *trivial*; no rules apply to it.

The search space is disjunctive, in that above some proof lines there will be multiple applicable rules. We use a dashed line to represent disjunctive branching in a search tree. Given a tree $\Pi$, let $\Pi^A$ be the tree obtained by adding $A$ into the context of each line and deleting rule applications above trivial lines. If $\mathcal{P}$ is a set of trees, let $\mathcal{P}^A$ result from applying the operation to each member.

Let $\Gamma \vDash^{\mathrm{i}} C$ be an i-line, let $\mathcal{P}$ be the set of subtrees rooted at rules immediately above $\Gamma \vdash C$ which do not contain any leaves of the form $A \vee B\ ?\ B$, and let $\mathcal{S}$ be the set of subtrees rooted at rules immediately above $\Gamma \vDash^{\mathrm{i}} C$ which do not contain any leaves of the form $A \vee B\ ?\ A$. The following transformation, i.e. tactic $\vee$E, may be applied whenever $\mathcal{P} \not\subseteq \mathcal{Q}$, $\mathcal{Q} \not\subseteq \mathcal{P}$, and $A \vee B \leq \Gamma$:

[2] $A \vee B\ ?\ A$ means ??? in one line?

[3] This means ??? in one line?



## 4 Implementing NIC proof search in Ω-ANTS

The rules of the NIC calculus are defined as tactics in ΩMEGA. Their side conditions as well as the overall proof search restrictions and heuristics are encoded into the specifications of the particular argument, command, and suggestion agents involved in our blackboard mechanism. The automate wrapper itself does not apply any calculus specific knowledge and the NIC specific proof search behavior is completely determined by the agents specifications.

*Encoding of structural application conditions and side conditions* We briefly sketch two agents working for the NIC-tactic $\wedge E_l$. We assume that the formal arguments of the command envoking $\wedge E_l$ are named 'conc' (for the conclusion line), 'conj' (for the premise line, i.e. the conjuction), and 'term' (referring to an additional term-parameter extending the presentation of $\wedge E_l$ in Section 3; this parameter is needed to realize backward applications of $\wedge E_l$ where the conjunction line to be introduced is unknown and has to be specified). In order to ease agent specifications we developed a partially declarative description language that includes the predicate (or function) of the particular agent in LISP code.

```
(agent~defmatrix NICTAC-AND-E-L
 (agents       ;; We define two agents for tactic 'NICTAC-AND-E-L': The first agent looks
  (s-predicate ;; for support lines which are appropriate candidates for parameter 'conj'
   (for conj)  ;; in dependence of some already given suggestion for conclusion line 'conc'.
   (uses conc) ;; Agent specification: 'conj' has to be an elimination line with a conjunct-
   (definition ;; ion as formula whose left conjunct is equal to 'conc'.
    (and (nic~elimination-p (:line conj)) (data~equal (left-conjunct conj) conc))))
  (c-predicate     ;; The second agent is looking for open lines which are appropriate can-
   (for conc term) ;; didates for parameter 'conc'. If successful this agent returns apart
   (multiple term) ;; from a suggestion for 'conc' suggestions for the parameter 'term'.
   (uses )         ;; This agent depends not on already given suggestions on the blackboard.
   (definition ;; Agent specification: Let 'termcandidates' be all conjunctions among the
    ...        ;; strictly positive subformulas of the support lines of open line  'conc'
               ;; which have a left conjunct equal to the formula of 'conc'. If there are any
               ;; such termcandidates, they are suggested together with the line 'conc'.
  (s-predicate (for term) (uses conc) ....
   )))))
```

*Encoding of rule/tactic preferences and global search strategies* The ordering of the suggestions being passed from the agent mechanism to the automate wrapper in each proof step determines the system's backtracking strategy. In the system suggestions are heuristically ordered on two layers. (i) The command agents monitoring the suggestion blackboards employ criteria like 'completeness' (how many parameters of the command are filled) and 'actuality' (what is the average age of the proof lines suggested as instantiations for the parameters) before passing the most promising one to the command blackboard. (ii) The suggestion agent monitoring the command blackboard employs (among other subordinated criteria) the following partial ordering of NIC tactics when passing the command suggestions to the automate wrapper[4]: $\{\rightarrow \text{ E}, \vee\text{E}, \wedge\text{E}\} > \{\rightarrow$

[4] Simultaneously they are also passed to the graphical user interface for interaction.

$\mathrm{I}, \wedge\mathrm{I}, \vee\mathrm{I}_{l,r}\} > \{\bot_{\mathbf{c}}\} > \{\vee\mathrm{E}_{l,r}\}$. Instead of a pure static ordering it is possible to specify dynamic ordering criteria taking the proof state at run time into account. In fact, we already make use of such dynamic criteria as the tactic $\vee$E is not suggested unless its applicability for particular lines is indicated by respective applications of the dummies $\vee\mathrm{E}_l$ and $\vee\mathrm{E}_r$ during the search process.

## 5 Conclusion and Implementation Notes

Ω-ANTS proposes a new system architecture for integrated interactive and automated theorem proving and its main idea is definitely not to provide another competitive automated theorem prover to the ATP community. Despite its current specialization on the NIC-calculus Ω-ANTS can be characterized as an open and easy extendible theorem proving approach with anytime character that is based on societies of concurrently working software agents associated with rules, tactics, methods, or external reasoners. As the system supports the redefinition of agents at run time, the user can dynamically influence the systems search behavior by adding, removing, or modifying single agents. If agent specifications for elaborate proof planning methods, other powerful calculi, or external reasoners are available, they can easily be added to the running system as well.

Further work includes the improvement of the interlocking of interactive and automated command executions without loosing important context and backtracking information in the automate wrapper (respective information stemming from intermediate interactions is currently not available to the automate wrapper). We will also experiment with extensions of the current calculus by other proof tactics. Furthermore, we want to fruitfully incorporate the resource adaptivity of the current suggestion mechanism (cf. [2]) into the automation. The propositional NIC-agents are currently tested and further tuned with examples from the TPTP-library.

Ω-ANTS is implemented as a tool for the ΩMEGA environment in Allegro Common Lisp using its multiprocessing facilities. It can be accessed via URL http://www.ags.uni-sb.de/~omega/www/oants.html and can be run from within ΩMEGA's graphical user interface web applet. When activating Ω-ANTS with the introduced propositional logic NIC rules we have a total of 59 concurrently working agents (for first-order NIC we currently experiment with ∼100 agents).

## References


1. C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In *Proc. of AIMSA'98*, LNAI 1480, pages 102–114. Springer, 1998.
2. C. Benzmüller and V. Sorge. Critical agents supporting interactive theorem proving. In *Proc. of EPIA'99*, LNAI 1695, pages 208–221. Springer, 1999.
3. J. Byrnes. *Proof Search and Normal Forms in Natural Deduction.* PhD thesis, Dept. of Philosophy, Carnegie Mellon University, Pittsburgh, PA 15213, USA, 1999.
4. The ΩMEGA Group. ΩMEGA: Towards a mathematical assistant. In *Proc. of CADE-14*, LNAI 1249, pages 252–255. Springer, 1997.
5. D. Prawitz. *Natural Deduction: A Proof-Theoretical Study.* Almqvist & Wiskell, Stockholm, 1965.