

Synthesizing Proof Planning Methods and Ω -Ants Agents from Mathematical Knowledge

Serge Autexier^{1,2} and Dominik Dietrich²

¹ German Research Center for Artificial Intelligence (DFKI GmbH), Saarbrücken, Germany, autexier@dfki.de

² FR 6.2 Informatik, Saarland University, Saarbrücken, Germany
{[autexier](mailto:autexier@ags.uni-sb.de)|[dodi](mailto:dodi@ags.uni-sb.de)}@ags.uni-sb.de

Abstract. In this paper we investigate how to extract proof procedural information contained in declarative representations of mathematical knowledge, such as axioms, definitions, lemmas and theorems (collectively called *assertions*) and how to effectively include it into automated proof search techniques. In the context of the proof planner MULTI and the agent-based reasoning system Ω -ANTS, we present techniques to automatically synthesize proof planning methods and Ω -ANTS-agents from assertions such that they can be *actively* used by these systems. This in turn enables a user to effectively use these systems without having to know the peculiarities of coding methods and agents.

1 Introduction

The development of the proof assistant system Ω MEGA is one of the major attempts to build an all encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. The search for a proof is usually conducted at a high level of granularity defined by *tactics* and *methods*. Automation of proof search at this 'abstract' level is realized in two components: The multi-strategy proof planner MULTI [13] and the resource-guided agent-based reasoning system Ω -ANTS [4, 15]. MULTI integrates a basic set of algorithms parameterized over strategic information. Among others, it includes a best-first proof planning algorithm that is parameterized over methods and control rules and which searches through the space of applicable methods by using the heuristic function defined by the provided control knowledge. The Ω -ANTS-system is based on encapsulations of calculus rules, tactics, external system calls and methods into pro-active agents which automatically check for their own applicability. Each of these agents is associated with a set of *argument agents* for formal arguments of the encapsulated procedure that compute possible instantiations for the associated formal argument subject to existing instantiations for other formal arguments.

The tactics and methods encode specific proof knowledge, such as, for instance, when and how to perform a proof by case analysis, to use a diagonalization argument, to call a computer algebra system or an automated theorem

prover, but also axioms, lemmas or theorems. The latter are typically contained in the mathematical theories but needed to be reformulated manually as tactics, methods and agents in order to enable MULTI and Ω -ANTS to *actively* use them during proof search. Hence a human user has to know the peculiarities of coding tactics, methods and agents for these systems in order to be able to make effective use of their automated proof search capabilities. In this paper we aim at remedying this situation. Based on a notion of *inferences* integrating the so far separated notions of tactics and methods, (1) we propose a technique to synthesize inferences from axioms, lemmas and theorems, (2) we define proof planning directly at the level of arbitrary, i.e. both the synthesized *and* the user-defined inferences, and (3) provide a mechanism to automatically generate an optimal set of argument agents from inferences.

The paper is organized as follows: In Sec. 2 we set the context of this work by describing the tactics, methods and Ω -ANTS-agents of the old Ω MEGA system. In Sec. 3 we introduce *inferences* as one major means for proof construction of the so-called task layer in the new Ω MEGA-system. In Sec. 4 we present a technique to synthesize *inferences* from axioms, definitions, lemmas, and theorems contained in mathematical theories. A procedure to determine the possible directions in which an inference can be applied is presented in Sec. 5 and the computation of optimal sets of Ω -ANTS-agents is given in Sec. 6. We discuss related works in Sec. 7 before concluding the paper with a summary of the results in Sec. 8.

2 Tactics, Methods, Ω -Ants

We briefly describe the context of the work presented in this paper by introducing tactics, methods and Ω -ANTS as they exist in the old Ω MEGA system. Throughout this section by Ω MEGA we mean the old Ω MEGA system [5].

Tactics. Ω MEGA provides the definition of tactics as a means for proof construction at an abstract level. These tactics are comparable to standard LCF-style tactics [12], that encode repeatedly occurring sequences of calculus steps combined by so-called *tacticals* such as *repeat*, *then*, *or*. Intuitively we can see a tactic as a program which performs a certain task. It is executed when the tactic is invoked and returns a verifiable proof object if it terminates. In Ω MEGA a tactic requires certain assumptions and open goals to be present in the current proof state in order to be applicable: these are, respectively, the premises P_1, \dots, P_k and conclusions C_1, \dots, C_n of the tactic. In practice, however, there are classes of tactics which essentially coincide, and only slightly differ in the premises and conclusions they require. The reason for this is that a tactic with input $P_1, \dots, P_n, C_1, \dots, C_k$ can often also be applied with input $P_1, \dots, P_{n-1}, C_1, \dots, C_k$ only, but then introduces P_n as additional subgoal. It is clear that all these cases could in principle be combined in one tactic, but this was too cumbersome to do in Ω MEGA's tactics specification formalism and hence was never done.

Methods. A method is a declarative specification of a tactic that describes how an application of the tactic modifies a given proof state. The underlying idea is to

be able to transform a proof state without the need to execute the tactic. Given such a specification, it must contain information which premises and conclusions are to be added to the proof state and which are to be removed during method application. In Ω MEGA premises and conclusions can be annotated with flags \oplus and \ominus : those annotated with \oplus will be added when applying the method, those annotated with \ominus shall be removed and those without annotations must be present before and after application of the method.

In general there is no guarantee that the specification really describes the tactic to which it belongs; hence methods can be unsound. To check the validity of a method proof step, the method must be expanded, that is the tactic attached to the method must be executed and return a verifiable proof object. Proof search using methods is called *proof planning* [6]. In contrast to other proof planning systems, such as λ -CLAM [14], Ω MEGA methods do not contain heuristic knowledge which could also be encoded in the specification.

Ω -ANTS. The Ω -ANTS-system has originally been developed to support a user in an interactive theorem proving environment by distributively searching via agents for the next possible proof steps [4, 15] and was later extended to a fully automated system. Conceptually the system consists of two kinds of agents: *command agents* and *argument agents*. Each command agent is associated to a tactic, method or calculus rule, uniformly called rule, and orders suggestions for the associated rule according to some heuristics. These suggestions are generated by a society of argument agents which are assigned to a command agent. From the perspective of this paper we are only interested in finding suggestions for a particular rule and thus only describe argument agents in more detail.

The goal of a society of argument agents consists in generating suggestions of how a particular rule can be applied in the current proof situation. Such a rule consists of premises, conclusions and a set of additional parameters, called arguments. Starting from the situation where no argument is instantiated, the agents stepwise instantiate these arguments, resulting in so-called *partial argument instantiations*. If sufficiently many arguments are instantiated, the rule can be applied. Intuitively an argument agent is responsible for one or more specific arguments – stored in the *goal set*. Given a set of already computed partial argument instantiations it checks whether it can add an instance for those arguments in its goal set. Usually an argument agent requires that certain formal arguments are already specified. These are stored in the so-called *dependency set*.

Discussion. Whereas in principle it is beneficial to have tools for interactive and automated proof search which can be combined to find a proof, for a Ω MEGA user this means that before starting a proof he has the burden to specify tactics for interactive proof construction, corresponding methods for the proof planner MULTI [13] and corresponding agents for proof construction with the Ω -ANTS-system. In particular he had to separately specify very similar tactics (and thus methods) as described above. Usually an essential subset of these tactics was a formulation of an axiom of the theory in which a proof was constructed. In this paper we show how the workload of a user can be drastically reduced: First, we

introduce the notion of an *inference* which unifies the previously separated notions of tactics and methods. In particular we develop a mechanism to synthesize inferences from axioms, lemmas and theorems. Furthermore we present mechanisms to generate from individual inferences all tactics within a tactic class and an optimal set of argument agents for each tactic.

3 Task Layer

The Ω MEGA-system is currently under re-development where, among others, the underlying natural deduction calculus is exchanged for the CORE-calculus [1]. The task layer [7] is an instance of the new proof datastructure (PDS) [2] and is the uniform proof construction interface used by both the human user and the automated proof search procedures MULTI and Ω -ANTS. The nodes of the PDS are annotated with *tasks*, which are Gentzen-style multi-conclusion sequents augmented by means to define multiple foci of attention on subformulas that are maintained during the proof. Each task is reduced to a possibly empty set of subtasks by one of the following proof construction steps: (1) the introduction of a proof sketch [17]³, (2) deep structural rules for weakening and decomposition of subformulas, (3) the application of a lemma that can be postulated on the fly, (4) the substitution of meta-variables, and (5) the application of an inference.

Due to the presence of meta-variables, substitutions and alternative proof steps, the task layer extends the PDS from [2] by a mechanism to deal with substitutions of the same meta-variable in alternative subproofs (see [7] for details).

The logic used in Ω MEGA is a simply-typed higher-order logic with arbitrary many base types, the usual β -reduction and η -expansion rules and we consider all terms to be always in $\beta\eta$ long normal form, which is unique up to renaming of bound variables (α -equal) (see [3] for details).

For the purpose of the paper we assume $T_{\Sigma, \mathcal{V}}^{\mathcal{X}}$ are the terms build over a signature Σ , typed variables \mathcal{V} and typed meta-variables \mathcal{X} ($\mathcal{V} \cap \mathcal{X} = \emptyset$) and $wff_{\Sigma, \mathcal{V}}^{\mathcal{X}}$ is the set of terms of boolean type, also denoted as *formulas*: as usual quantifiers and λ -binders can only bind variables and we require the formulas from $wff_{\Sigma, \mathcal{V}}^{\mathcal{X}}$ to contain no free variables.

Each subterm of a term t can be uniquely qualified by its position π in the term which we denote by $t_{|\pi}$. The type compliant replacement of a subterm $t_{|\pi}$ by a term s is denoted by $t_{|\pi \leftarrow s}$. The set of all positions is denoted by \mathcal{POS} and $\mathcal{POS}(t)$ denotes the set of all valid positions of the term t .

A *substitution* is a type preserving and idempotent function $\sigma : \mathcal{V} \cup \mathcal{X} \rightarrow T_{\Sigma, \mathcal{V}}^{\mathcal{X}}$ that is the identity function but for finitely many elements from $\mathcal{V} \cup \mathcal{X}$. This allows for a finite representation of σ as $\{\sigma(x_1)/x_1, \dots, \sigma(x_n)/x_n\}$ where $\sigma(y) = y$ if $\forall 1 \leq i \leq n, y \neq x_i$. The *domain* of σ is $dom(\sigma) := \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ and we denote by *meta-variable substitution* those substitutions σ where $dom(\sigma) \subset \mathcal{X}$. As usual we do not distinguish between a substitution and its homomorphic extension to terms.

³ In the old Ω MEGA this was realized by using so-called *Island*-methods.

Each subformula occurs either negatively or positively in a formula and allows to assign a positive or negative polarity to the subformula subject to the polarity of the entire formula. Following [10], polarized propositional formulas can be classified into α -type and β -type formulas. We say that two subformulas F and G of some polarized formula are α -related (resp. β -related), if the smallest subformula containing both F and G is of type α (resp. β).

For the purpose of this paper we consider tasks as multi-conclusion sequents $F_1, \dots, F_k \vdash G_1, \dots, G_l$, where F_i and G_j are from $wff_{\Sigma, \mathcal{V}}^{\mathcal{X}}$. The notions of substitution and positions of formulas carry over to sequents in the obvious way, and we define the F_i to have negative polarity and the G_j to have positive polarity.

3.1 Inferences

Intuitively, an *inference* is a proof step with multiple premises and conclusions augmented by 1) a possibly empty set of hypothesis for each premise, 2) a set of *application conditions* that must be fulfilled upon inference application, and 3) a set of *outline functions* that can compute the values of premises and conclusions out of values of other premises and conclusions, and 4) an *expansion function* that refines the abstract inference step. Each premise and conclusion consists of a unique name and a formula scheme from $wff_{\Sigma, \mathcal{V}}^{\mathcal{X}}$. Note that we employ the term *inference* in its general meaning: Take in that sense, an inference can be either valid or invalid in contrast to the formal logic notion of an *inference rule*.

Additional information needed in the application conditions or the outline functions, such as, for instance, the position of a subterm or the instance of some non-boolean meta-variable, can be specified by additional *parameters* to the inference. Since this can be arbitrary information, we refrain from a formal definition, but assume that we can check the admissibility of a substitution of the formal parameters. The parameters of an inference, the names of premises and conclusions and the meta-variables that occur in the associated formulas form the *variables of the inference*.

Definition 1 (Inference Variables & Inference Substitutions). *The pairwise disjoint sets \mathcal{P} of parameter variables, \mathcal{N} of names for premises and conclusions and \mathcal{X} of meta-variables are the inference variables. Let further \mathbf{Val} denote the possible values for parameters and T be a task. An inference substitution wrt. T is a triple $\sigma := \langle \sigma_{\mathcal{P}}, \sigma_{\mathcal{N}}, \sigma_{\mathcal{X}} \rangle$, where (1) $\sigma_{\mathcal{P}} : \mathcal{P} \rightarrow \mathbf{Val} \cup \{\perp\}$ is an admissible parameter substitution, (2) $\sigma_{\mathcal{N}} : \mathcal{N} \rightarrow \mathcal{POS}(T) \cup wff_{\Sigma, \mathcal{V}}^{\mathcal{X}} \cup \{\perp\}$ is a name substitution, and (3) $\sigma_{\mathcal{X}}$ is a meta-variable substitution.*

The domain of the parameter substitutions $dom(\sigma_{\mathcal{P}})$ is the largest subset of \mathcal{P} on which the value of $\sigma_{\mathcal{P}}$ is not \perp . Analogously we define $dom(\sigma_{\mathcal{N}})$ and $dom(\sigma_{\mathcal{X}})$. Furthermore, we define $\sigma^{\#}$ to denote the function which returns $\sigma^{\#}(i) := \sigma_{\mathcal{X}}(T|_{\sigma_{\mathcal{N}}(i)})$ if $i \in \mathcal{N}$ and $\sigma_{\mathcal{N}}(i) \in \mathcal{POS}(T)$, and otherwise behaves like σ .

We say that an inference substitution $\langle \sigma_{\mathcal{P}}, \sigma_{\mathcal{N}}, \sigma_{\mathcal{X}} \rangle$ is *more general* than the inference substitution $\langle \tau_{\mathcal{P}}, \tau_{\mathcal{N}}, \tau_{\mathcal{X}} \rangle$, iff (1) there exists a substitution ρ such that

$$\frac{p_1 : F \quad p_2 : U = V}{c : G} \text{subst-}m(\pi)$$

Appl. Cond.: $(F|_{\pi} = U \wedge G|_{\pi \leftarrow V} = F) \vee (G|_{\pi} = U \wedge F|_{\pi \leftarrow V} = G)$
Outline: $\langle c, \text{compute-subst-}m(p_1, p_2, \pi) \rangle$
 $\langle p_1, \text{compute-subst-}m(p_2, c, \pi) \rangle$
 $\langle \pi, \text{compute-pos}(p_1, p_2) \rangle$
 $\langle \pi, \text{compute-pos}(p_2, c) \rangle$

Fig. 2. Inference *subst- m*

c with formula scheme G , and one parameter π . It represents the deduction step that if we are given a formula F in which at position π the term U occurs and we are given that $U = V$, then we can deduce the formula G which equals F except that U is replaced by V . The outline functions can be used to compute the conclusion formula c , given p_1, p_2 , and π or to compute the formula p_1 , given c, p_2 , and π , or to compute the position π at which the replacement can be performed. Note that there are two outline functions for computing π .

Definition 4 (Admissible and Fully Specified Inference Substitutions).
 Assume an inference $\mathbf{I} := \langle P, C, \text{Hyp}, \text{FS}, \Omega, P(i_k^0), \text{OF}, \text{Exp} \rangle$, an inference substitution σ wrt. some task T . We say that σ is admissible for \mathbf{I} and T iff (1) for all $p \in P$ and $c \in C$ such that $\sigma(p), \sigma(c) \in \text{POS}(T)$ $\sigma(p)$ and $\sigma(c)$ are α -related positions in T , (2) for all $c, c' \in C$ such that $\sigma(c), \sigma(c') \in \text{POS}(T)$ $\sigma(c)$ and $\sigma(c')$ are β -related positions in T (3) for all $i \in (P \cup C) \cap \text{dom}(\sigma)$ it holds (3.1) if $i \in P$ and $\sigma(i) \in \text{POS}(T)$ then $\sigma(i)$ is a negative position in T ; (3.2) if $i \in C$ and $\sigma(i) \in \text{POS}(T)$ then $\sigma(i)$ must be a positive position in T ; (3.3) if $\sigma(i) \in \text{wff}_{\Sigma, \mathcal{V}}^x$, then $\sigma(i) = \sigma(\text{FS}(i))$ must hold.

We say that σ is fully specified for \mathbf{I} and T iff it is admissible for \mathbf{I} and T and $P \cup C \subseteq \text{dom}(\sigma)$. Otherwise we say σ is partial.

4 Synthesizing Inferences from Mathematical Knowledge

In the old Ω MEGA-system the knowledge contained in the mathematical theories was not automatically available to the proof planner or the Ω -ANTS-system. In order to make them available for these systems, they had to be specified manually as methods or tactics and agents. In this section we present a mechanism that allows to compute a set of inferences for arbitrary formulas. The intuition is as follows: Given the following axiom which is a part of the definition of addition on natural numbers

$$\forall x, y. y > 0 \Rightarrow x + y = s(x + p(y)) \quad (1)$$

where s and p are the successor respectively the predecessor functions of natural numbers. We want it to result in the inference rules

$$\frac{P_1 : Y > 0 \quad P_2 : H(X + Y)}{C : H(s(X + p(Y)))} \quad \frac{P_1 : Y > 0 \quad P_2 : H(s(X + p(Y)))}{C : H(X + Y)} \quad (2)$$

Application Condition: –

Application Condition: –

$$\begin{array}{c}
\frac{R \star \mathcal{C}; \Gamma \vdash_R A \Leftrightarrow B}{R \star \mathcal{C}; \Gamma \vdash_R A \Rightarrow B \star \mathcal{C}; \Gamma \vdash_R B \Rightarrow A} \Leftrightarrow^E \\
\frac{R \star \mathcal{C}; \Gamma \vdash_R s = t}{R \star \mathcal{C}; \Gamma \vdash_R X(s) \Rightarrow X(t) \star \mathcal{C}; \Gamma \vdash_R X(t) \Rightarrow X(s)} =^E \\
\text{where } X \text{ new wrt. } \mathcal{C}; \Gamma \vdash_R s = t \\
\frac{R \star \mathcal{C}; \Gamma \vdash_R A \Rightarrow B}{R \star \mathcal{C}; \Gamma, \Box A \vdash_R B} \Rightarrow^E \\
\frac{R \star \mathcal{C}; \Gamma \vdash_R A \wedge B}{R \star \mathcal{C}; \Gamma \vdash_R A \star \mathcal{C}; \Gamma \vdash_R B} \wedge^E \\
\frac{R \star \mathcal{C}; \Gamma \vdash_R \forall x A}{R \star \mathcal{C}; \Gamma \vdash_R A[X/x]} \forall^E \\
\text{where } X \text{ new wrt. } \mathcal{C}; \Gamma \vdash_R \forall x A \\
\frac{R \star \mathcal{C}; \Gamma \vdash_R \exists x A}{R \star \mathcal{C}, c \notin \Gamma, \exists x A; \Gamma \vdash_R A[c/x]} \exists^E
\end{array}
\left|
\begin{array}{c}
\frac{R \star \mathcal{C}; \Gamma, [\Delta] A \Rightarrow B \vdash_R F}{R \star \mathcal{C}; \Gamma, [\Delta, A] B \vdash_R F} \Rightarrow^I \\
\frac{R \star \mathcal{C}; \Gamma, [\Delta] A \wedge B \vdash_R F}{R \star \mathcal{C}; \Gamma, [\Delta] A, [\Delta] B \vdash_R F} \wedge^I \\
\frac{R \star \mathcal{C}; \Gamma, [\Delta] A \vee B \vdash_R F}{R \star \mathcal{C}; \Gamma, [\Delta] A \vdash_R F \star \mathcal{C}; \Gamma, [\Delta] B \vdash_R F} \vee^I \\
\frac{R \star \mathcal{C}; \Gamma, [\Delta] \forall x A \vdash_R F}{R \star \mathcal{C}, c \notin [\Delta] \forall x A; \Gamma, [\Delta] A[c/x] \vdash_R F} \forall^I \\
\frac{R \star \mathcal{C}; \Gamma, [\Delta] \exists x A \vdash_R F}{R \star \mathcal{C}; \Gamma, [\Delta] A[X/x] \vdash_R F} \exists^I \\
\text{where } X \text{ new wrt. } \mathcal{C}; \Gamma, [\Delta] \exists x A \vdash_R F \\
\frac{R \star \mathcal{C}; \Gamma, [\Delta, A \wedge B] G \vdash_R F}{R \star \mathcal{C}; \Gamma, [\Delta, A, B] G \vdash_R F} \wedge^H
\end{array}
\right.$$

Fig. 3. Sets of Rules to compute conclusions & premises

where H, X, Y are meta-variables. Similarly, the following direction of the equivalence of the definition of the limit of a function

$$\begin{aligned}
\forall f, a, l. \forall \epsilon. \epsilon > 0 \Rightarrow \exists \delta. \delta > 0 \Rightarrow \forall x. (0 < |x - a| \wedge |x - a| < \delta) \Rightarrow |f(x) - l| < \epsilon \\
\Rightarrow \lim_a f = l
\end{aligned} \tag{3}$$

should give us the inference

$$\frac{[\epsilon > 0, D > 0, 0 < |x - A|, |x - A| < D] \quad \begin{array}{c} \vdots \\ P : |F(x) - L| < \epsilon \end{array}}{C : \lim_A F = L} \tag{4}$$

Application Condition: $\text{EV}(\epsilon, \{F, A, L\}) \wedge \text{EV}(x, \{F, A, L, D\})$

Parameters: ϵ, x

where F, A, L, D are meta-variables, ϵ and x are parameters to the rule and $\text{EV}(x, \{F, A, L, D\})$ is the application condition requiring that the parameter x should not occur in the instances of $\{F, A, L, D\}$.

Our mechanism is inspired by the generalized natural deduction procedure proposed by Wack [16]. It determines the application directions of formulas by following the introduction and elimination rule structure of a natural deduction (ND) calculus [11]. Given a formula, in a first phase the ND elimination rules are exhaustively applied to that formula and where we eliminate equivalences in the obvious way and handle equality via Leibniz' definition of equality. While applying the rules, we collect the *Eigenvariable* conditions which results in a set of inference descriptions with *Eigenvariable* conditions. In a second phase the premises of the inference descriptions are simplified by exhaustively applying ND introduction rules to the premises as well as to possible hypotheses obtained for the premises in that phase. The two sets of rules are given in Fig. 3: We use

$R \star x$ to denote $R \cup \{x\}$ and the notation $\mathcal{C}; \Gamma \vdash_R C$ for inference descriptions stating that the conclusion C can be derived from the hypotheses in Γ if the conditions in \mathcal{C} are respected. The premises in Γ are of the form $[\mathcal{H}]P$ stating that in order to show the premise P we can assume the hypotheses \mathcal{H} . The *Eigenvariable* conditions collected in \mathcal{C} are of the form “ y new wrt. S ”, where y is the Eigenvariable and S is a list of constants and meta-variables in which y shall not occur (including the symbols in the meta-variable substitutions). Checking these conditions by using the predicate $\text{EV}(y, S)$ s, an inference description $\text{EV}(y_1, S_1), \dots, \text{EV}(y_m, S_m); [\mathcal{H}_1]P_1, \dots, [\mathcal{H}_n]P_n \vdash_R C$ gives rise to the inference

$$\frac{
 \begin{array}{ccc}
 [\mathcal{H}_1] & & [\mathcal{H}_n] \\
 \vdots & & \vdots \\
 \dot{P}_1 & \dots & \dot{P}_n
 \end{array}
 }{C} \text{Parameters } (y_1, \dots, y_n)$$

Application Condition: $\text{EV}(y_1, S_1) \wedge \dots \wedge \text{EV}(y_m, S_m)$

The elimination rules for decomposition of the conclusions in the first phase are on the left-hand side and the introduction rules for the premises and the single elimination rule for the hypotheses are on the right-hand side.

It is obvious to see that both sets of rules are terminating and confluent (up to the ordering of the premises and their hypotheses, and the renaming of introduced Eigenvariables and meta-variables). We illustrate the procedure by considering the axioms (1) and (3): For (1) the initial inference description is $.; \vdash_R \forall x, y. y \neq 0 \Rightarrow x + y = s(x + p(y))$ and saturation using the elimination rules yields the sets $\{.; \Box Y > 0, \Box H(X + Y) \vdash_R H(s(X + p(Y)))\}; \Box Y > 0, \Box H(s(X + p(Y))) \vdash_R H(X + Y)\}$. None of the rules from the second set of rules is applicable and hence we obtain the two inferences from (2).

The initial inference description for (3) is $.; \vdash_R \forall f, a, l. (\forall \epsilon. \epsilon > 0 \Rightarrow \exists \delta. \delta > 0 \Rightarrow \forall x. (0 < |x - a| \wedge |x - a| < \delta) \Rightarrow |f(x) - l| < \epsilon) \Rightarrow \lim_a f = l$ and the elimination rules terminate yield the singleton $\{.; \Box \forall \epsilon. \epsilon > 0 \Rightarrow \exists \delta. \delta > 0 \Rightarrow \forall x. (0 < |x - A| \wedge |x - A| < \delta) \Rightarrow |F(x) - L| < \epsilon \vdash_R \lim_A F = L\}$. The second set of rules yields the singleton set

$$\left\{ \text{EV}(\epsilon, \{F, A, L\}), \text{EV}(x, \{F, A, L, D\}); \left[\epsilon > 0, D > 0, 0 < |x - A|, |x - A| < D \mid F(x) - L < \epsilon \vdash_R \lim_A F = L \right] \right\}.$$

from which we obtain the inference (4).

5 Partial Argument Instantiations

The problem of determining the different possibilities to apply a given inference on some task consists of finding all fully specified inference substitution. Typically, some of the parameters of the inference are instantiated as well as possibly some of the formal arguments. Hence, the process starts with a partial inference

substitution and we have to find the values for the non-instantiated variables of the inference. During the process these variables are assigned either an position of the task or a formula. In order to make the stages of the process explicit, we define the notion of *partial argument instantiation*, which is an adaptation of the notion from [4, 15] to the new inferences presented here.

Definition 5 (Partial Argument Instantiation). *Let \mathbf{I} be an inference with formal arguments a_1, \dots, a_n and T a task. Any inference substitution $PAI_{\mathbf{I}}^T := \langle \sigma_{\mathcal{P}}, \sigma_{\mathcal{N}}, \sigma_{\mathcal{X}} \rangle$ admissible for \mathbf{I} and T is a partial argument instantiation (PAI). A PAI $PAI_{\mathbf{I}}^T$ is called empty, iff $dom(PAI_{\mathbf{I}}^T) = \emptyset$; it is called initial iff for all $1 \leq j \leq n$ holds $\sigma_{\mathcal{N}}(a_j) \notin \text{wff}_{\Sigma, \mathcal{V}}^{\mathcal{X}}$ and for all $p \in \mathcal{P}$ holds $\sigma_{\mathcal{P}}(p) = \perp$; finally we say it is complete iff there is sequence of outline functions $\langle V_i, f(\mathbf{I}_{k_i}^k), 0 \leq k \leq m \rangle$ such that $PAI_{\mathbf{I}}^T \oplus \langle v_1, f(\mathbf{i}_{k_1}^1) \rangle \oplus \dots \oplus \langle v_m, f(\mathbf{i}_{k_m}^m) \rangle$ is a fully specified inference substitution wrt. \mathbf{I} and T .*

Example 1. Let us reconsider the inference *subst-m* from Fig. 2, which we want to apply to the task $T: 2 * 3 = 6 \vdash 2 * 3 < 7$. Then the substitution $pai_1 = \langle \emptyset, \{c \mapsto (10)\}, \emptyset \rangle$ is a PAI for the inference *subst-m*, where (10) denotes the position of $2 * 3 < 7$ in the task. As no outline function has been invoked so far pai_1 is initial. It is not complete as there are no outline functions to compute π given only c ; thus p_1, p_2 can also not be computed.

The extension of a partial argument instantiation $PAI_{\mathbf{I}}^T$ consists of an assignment of values to variables of the inference that are not in the domain of $PAI_{\mathbf{I}}^T$. We are only interested in those extensions where at least one not yet instantiated *formal argument* is assigned a new value.

Definition 6 (Partial Argument Instantiation Update). *Let \mathbf{I} be an inference, T be a task, $PAI_{\mathbf{I}}^T, PAI_{\mathbf{I}}^{\prime T}$ be partial argument instantiations for \mathbf{I} with respect to T . Then $PAI_{\mathbf{I}}^{\prime T}$ is a partial argument update of $PAI_{\mathbf{I}}^T$ iff (1) $PAI_{\mathbf{I}}^{\prime T}$ is more general than $PAI_{\mathbf{I}}^T$ and (2) there is at least one formal argument of \mathbf{I} in $dom(PAI_{\mathbf{I}}^{\prime T}) \setminus dom(PAI_{\mathbf{I}}^T)$.*

There are two possibilities to perform a partial argument instantiation update: (1) assigning a task position to a formal argument or (2) assigning a term to a formal argument. The first kind of updates involves search for possible positions in the task while respecting already introduced bindings. The second kind of updates involves no search in practice and is performed by using the outline functions which require that already some arguments are instantiated.

Thus we can divide the updating process into two phases: In the first phase we allow only updates of the first kind and in the second phase only updates of the second form. The underlying idea is that we try to use as many derived knowledge as possible and then decide whether this knowledge suffices for the inference to be applied. The knowledge in an initial PAI suffices for the outline functions to compute all other values, if the initial PAI is complete. Each initial and complete PAI determines one way the inference can be applied.

Example 2. If we add an instantiation for the argument p_2 in pai_1 we obtain a new PAI $pai_2 = \langle \emptyset, \{p_2 \mapsto (00), c \mapsto (10)\}, \emptyset \rangle$, where (00) denotes the position of the formula $2 * 3 = 6$ in our task T . p_2 is a PAI-update of p_1 . It is complete, as we can invoke the outline functions to first obtain π and then to obtain p_1 . Note that this does not mean that we have to invoke the outline functions, but we can try to instantiate further arguments by formulas of our task.

Abstracting from the concrete values the formal parameters are instantiated by these PAIs, we obtain a general descriptions of the application directions of the inferences.

The determination of the application direction in turn can be used (1) for planning as we know what the possible effects of an inference application are, and (2) for generating agents as it is reasonable that the agents must be able to construct partial argument instantiations for all application directions.

In the remainder of this section we formalize the notion of application direction by defining *PAI-statuses* that represent the status of a PAI rather than the concrete values.

Definition 7 (PAI-Status, PAI-Status defined by a PAI). *Let \mathbf{I} be an inference with formal arguments \mathbf{A} and T a task. A PAI-status $\mathcal{S}_{\mathbf{I}}$ of \mathbf{I} is a function $\mathcal{S}_{\mathbf{I}} : \mathbf{A} \rightarrow \{TERM, POS, \perp\}$. Its domain is $\text{dom}(\mathcal{S}_{\mathbf{I}}) := \{a \mid \mathcal{S}_{\mathbf{I}}(a) \neq \perp\}$. A PAI $PAI_{\mathbf{I}}^T$ of \mathbf{I} belongs to some PAI-status $\mathcal{S}_{\mathbf{I}}$ iff for all $a \in \mathbf{A}$ it holds (\perp) if $PAI_{\mathbf{I}}^T(a) = \perp$ then $\mathcal{S}_{\mathbf{I}}(a) = \perp$, (POS) if $PAI_{\mathbf{I}}^T(a) \in \mathcal{POS}$ then $\mathcal{S}_{\mathbf{I}}(a) = POS$, or $(TERM)$ if $PAI_{\mathbf{I}}^T \in \text{wff}_{\Sigma, \mathcal{V}}^x$ then $\mathcal{S}_{\mathbf{I}}(a) = TERM$.*

We say that two PAIs are *equivalent*, written $PAI_{\mathbf{I}}^{(T)} \sim PAI_{\mathbf{I}}^{(T')}$, if they belong to the same PAI status.

Example 3. As an example consider the PAI pai_2 . The status defined by this PAI is a function $f : \{p_1, p_2, c\} \rightarrow \{TERM, POS, \perp\}$ with $f(p_1) = \perp$, $f(p_2) = POS$, and $f(c) = POS$. Suppose we are given another task $T_2 = 2 + 2 = 4 \vdash A \subset B, 2+2 = 1+1+1+1$ with a PAI for *subst-m* $pai_3 = \langle \emptyset, \{p_1 \mapsto (00), c \mapsto (11)\}, \emptyset \rangle$. Then pai_3 and pai_2 have the same status.

The notions of updating a PAI, as well as initial and complete PAIs carry over to PAI-statuses in a straightforward manner:

Definition 8 (PAI-Status Update). *Let $\mathcal{S}, \mathcal{S}'$ be PAI-statuses of an inference \mathbf{I} . \mathcal{S}' is called PAI-status-update iff (1) $\forall x \in \text{dom}(\mathcal{S}), \mathcal{S}'(x) = \mathcal{S}(x)$ and (2) there occurs at least one formal argument of \mathbf{I} in $\text{dom}(\mathcal{S}') \setminus \text{dom}(\mathcal{S})$.*

Note that the PAI-status update relationship is a partial order on PAI-statuses, which we denote by $<_{\mathcal{S}}$.

Consequently, the status of pai_2 is a PAI-status update of the status of pai_1 .

Definition 9 (Initial, Empty, Complete, and Full PAI-Statuses). *Let \mathcal{S} be a PAI-status of an inference \mathbf{I} with formal arguments \mathbf{A} . Let further \mathcal{S}^{-1} denote the inverse image of \mathcal{S} defined by $\mathcal{S}^{-1}(x) := \{a \in \mathbf{A} \mid \mathcal{S}(a) = x\}$. We*

say that \mathcal{S} is (1) initial iff $\mathcal{S}^{-1}(TERM) = \emptyset$, (2) it is empty iff $\mathcal{S}^{-1}(POS) = \mathcal{S}^{-1}(TERM) = \emptyset$, (3) it is complete iff there exists a complete PAI $PAI_{\mathbf{I}}^{(T)} \in \mathcal{S}_{\mathbf{I}}$, and (4) it is full iff there exists a full PAI $PAI_{\mathbf{I}}^{(T)} \in \mathcal{S}_{\mathbf{I}}$.

Notational Convention. Given an inference \mathbf{I} , the empty PAI-status of \mathbf{I} is denoted by \mathcal{S}^{\emptyset} . We agree to denote an initial PAI-status \mathcal{S} where $\mathcal{S}^{-1}(POS) = \{a_1, \dots, a_n\}$ by $\langle a_1, \dots, a_n \rangle$.

Definition 10 (Application Directions). Let \mathbf{I} be an inference. The initial and complete PAI-statuses of \mathbf{I} are the application directions $\mathcal{AD}_{\mathbf{I}}$ of \mathbf{I} .

Example 4. As an example consider again the inference *subst-m*: it has the 3 application directions $\langle p_1, p_2, c \rangle$, $\langle p_1, p_2 \rangle$, and $\langle p_2, c \rangle$.

To check whether or not an initial \mathcal{S} is an application direction, i.e. is complete, we construct a so-called *PAI-status completion tree* for the \mathcal{S} we want to test. Each node of the tree is labeled with a PAI-status and we add an edge from a PAI-status \mathcal{S}' to some \mathcal{S}'' , if there is an outline function $of := \langle v, f(i_1, \dots, i_n) \rangle$ which is applicable on \mathcal{S} , i.e. $\{i_1, \dots, i_n\} \subseteq dom(\mathcal{S})$ and $\mathcal{S}(v) = \perp$. We recursively construct that tree starting with the given \mathcal{S} , and finally check if there is at least one *full PAI-status* among the leaf-nodes of the tree. If so, then \mathcal{S} is complete, i.e. is an application direction, and otherwise not. If $\mathcal{S}^{-1}(a) = TERM$, a is annotated with \oplus , otherwise if $a \in C$ with \ominus , if $a \in P$ it remains unannotated.

6 Generating Agents

Given a specification of an inference, we want to create a set of argument agents for the Ω -ANTS-system which provide the user with suggestions how the formal arguments of the inference can be instantiated. As the agents must cooperate to find instantiations, the benefit of a single agent cannot be assessed, rather we have to see the agents as a unit. These units are “fragile” in the sense that removing one agent can result in a non-operational unit that cannot produce useful suggestions or any suggestions at all. Hence we must chose a sufficiently large set of agents such that for each agent there is another agent which produces partial argument instantiations required by the agent. On the other hand each agent consumes runtime. If we created all possible agents the system performance would deteriorate as it would take too much time to create a suggestion. Hence we have to construct a set of agents which forms an operational unit and is as small as possible.

Intuitively a unit of agents for an inference reads partial argument instantiations from a blackboard. If a single agent realizes that it can perform an update, it performs that update and writes the new partial argument instantiation onto the blackboard, such that it can be further updated by the other agents of the unit. We aim at a unit of agents that is able to find all application directions of an inference, provided instances of these application directions can be built with respect to the task to which the agents are applied.

Agent Creation Graph. The agent creation graph is composed of nodes labeled with initial *PAI*-statuses and edges between these nodes, representing *PAI*-status updates. We only encode updates in the graph which update exactly one formal argument.

Definition 11 (Agent Creation Graph). *Let \mathbf{I} be an inference and $N = \{\mathcal{S}_\mathbf{I}^1, \dots, \mathcal{S}_\mathbf{I}^n\}$ be the set of all initial *PAI*-statuses of \mathbf{I} . Let $(\mathcal{S}_\mathbf{I}^i, \mathcal{S}_\mathbf{I}^j) \in E$ iff $\mathcal{S}_\mathbf{I}^j$ is a *PAI* status update of $\mathcal{S}_\mathbf{I}^i$ and $|\text{dom}(\mathcal{S}_\mathbf{I}^i)| + 1 = |\text{dom}(\mathcal{S}_\mathbf{I}^j)|$. Then $G = \langle N, E \rangle$ is called agent creation graph where each edge is labeled with $\oplus A$, where A is the formal argument additionally instantiated in $\mathcal{S}_\mathbf{I}^j$.*

An argument agent \mathfrak{A} consists of a dependency set $\mathcal{D}_\mathfrak{A}$ of formal arguments and a goal set $\mathcal{G}_\mathfrak{A}$ of formal arguments. Hence each edge in an agent creation graph corresponds to exactly one argument agent with dependency set those formal arguments instantiated in the source *PAI*-status and goal set that formal argument which is additionally instantiated in the target *PAI*-status. Hence we can associate to a given set of argument agents \mathfrak{K} that subgraph of the agent creation graph covered by these argument agents. This graph is called *actual agent graph*.

Definition 12 (Actual Agent Graph). *Let $G = \langle N, E \rangle$ be an agent creation graph and \mathfrak{K} a set of agents. The smallest subgraph of G that contains for each $\mathfrak{A} \in \mathfrak{K}$ an edge $(\mathcal{S}, \mathcal{S}')$ such that $\text{dom}(\mathcal{S}) = \mathcal{D}_\mathfrak{A}$ and $\text{dom}(\mathcal{S}') \setminus \text{dom}(\mathcal{S}) = \mathcal{G}_\mathfrak{A}$ is the actual agent graph of \mathfrak{K} .*

In an agent creation graph, those *PAI*-statuses which are complete are those which must be reachable from the root node of the graph, since they are the states which can be transformed in a fully specified *PAI*-status, in which the associated inference is applicable. In order to have a minimal set of agents \mathfrak{K} we require that in the actual agent graph of \mathfrak{K} there is *exactly one* path from the root node to each complete *PAI*-status. From this we can easily derive the following algorithm to create agent units for a given inference.

Algorithm to Create an Agent Unit. Given an inference \mathbf{I} and the complete initial *PAI*-statuses partially ordered wrt. $<_{\mathcal{S}}$ (Def. 8). The algorithm creates a unit of argument agents for \mathbf{I} such that the given complete *PAI*-statuses are all reachable from the empty *PAI*-status.

Initially, the set of agents is empty and the actual agent graph contains only the empty *PAI*-status. The algorithm recurses over the given list of complete *PAI*-statuses and in each iteration selects the first element \mathcal{S} of the list of *PAI*-statuses. Since they are ordered by $<_{\mathcal{S}}$, \mathcal{S} is not yet reachable in the actual agent graph. It then calculates the shortest paths⁴ in the agent creation graph to \mathcal{S} from all *PAI*-statuses that are already in the actual agent graph. It selects a path that has minimal distance to \mathcal{S} and adds agents for each edge on that path to the set of agents. As the list of statuses is ordered and always the minimal path is added the algorithm produces an optimal set of agents.

⁴ Using, for instance, Dijkstra's algorithm [8].

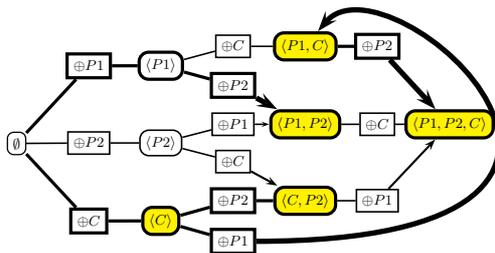


Fig. 4. Agent Creation Graph

Figure 4 shows an agent creation graph for an inference \mathbf{I} which has two premises $P1$ and $P2$ and one conclusion C . Suppose that the complete PAI-statuses of \mathbf{I} are $\langle C \rangle$, $\langle C, P2 \rangle$, $\langle P1, P2 \rangle$, $\langle P1, C \rangle$, and $\langle P1, P2, C \rangle$. The complete PAI-statuses are indicated in gray and the actual agent graph constructed by our algorithm is indicated by the bold edges and nodes. It illustrates that the agents optimally cooperate to find partial argument instantiations. A comparison in [7] of some automatically generated units of argument agents with manually specified argument agents shows that both are almost identical.

7 Related Work

Closest to the presented style of operationalizing mathematical knowledge are the *macetes* in the IMPS-system [9] and the *replacement rules* of the CORE-calculus [1]. Macetes only work with pure universal formulas in prenex normal-form and require the formulas to be given in the right format; the conditions are not further decomposed and equations (resp. equivalences) are only applied from left to right. Our mechanism in turn is less flexible than the synthesis of replacement rules: our rules require that negations occur only on atoms inside the formulas in order to have only literals as premises while the CORE mechanisms would not require this restricted format. Adding this feature to our rules would not be a problem, but we refrained from doing so, because in the cases exploiting that feature it would be difficult for a human user to recognize the original formula from the generated inferences. Consider as an example the formula $\neg(A \wedge (B \Rightarrow C))$ and the then generated inferences $.; \Box A \vdash_R B$ and $.; \Box A \vdash_R \neg C$. Furthermore, the result of the synthesis must meet the format of inferences and, hence, except for conjunctions, we cannot allow for rules that further decompose the hypotheses of premises.

8 Conclusion

We have presented an approach to extract proof procedural information from declarative representations of mathematical knowledge. It is based on a new notion of an inference and techniques to determine all sensible application directions and argument agents of inferences. We adapted a mechanism from [16]

to automatically synthesize inferences from assertions, which enables the proof-planner MULTI and the agent-based reasoning system Ω -ANTS to actively use knowledge contained in mathematical theories and to effectively assist the user even if he does not know the peculiarities of how to condition knowledge for these systems.

References

1. S. Autexier. The CORE calculus. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, July 2005. Springer.
2. S. Autexier, Chr. Benzmüller, D. Dietrich, A. Meier, and C.-P. Wirth. A generic modular data structure for proof attempts alternating on ideas and granularity. In M. Kohlhase, editor, *Proceedings of MKM'05*, LNAI 3863, IUB Bremen, Germany, January 2006. Springer.
3. H. Barendregt. *The λ -Calculus - Its Syntax and Semantics*. North Holland, 1984.
4. Chr. Benzmüller and V. Sorge. Ω -ANTS – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Proceedings of Calculemus-2000*, St. Andrews, UK, 6–7 August 2001. AK Peters.
5. Christoph Benzmüller, Armin Fiedler, Andreas Meier, Martin Pollet, and Jörg Siekmann. Omega. In *The seventeen provers of the world*, number 3600 in LNAI, pages 127–141, 2006.
6. Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE-88)*, LNAI, pages 111–120. Springer, 1988.
7. D. Dietrich. The task-layer of the Ω MEGA system. Diploma thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2006.
8. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
9. W. Farmer, J. Guttman, and F. J. Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11, 1993.
10. M. Fitting. *First-Order Logic and Automated Theorem Proving / 2nd Edition*. Springer-Verlag New York Inc., 1996. ISBN 0-387-94593-8.
11. G. Gentzen. *The Collected Papers of Gerhard Gentzen (1934-1938)*. Edited by Szabo, M. E., North Holland, Amsterdam, 1969.
12. M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF – A mechanised logic of computation*. Springer Verlag, 1979. LNCS 78.
13. A. Meier and E. Melis. MULTI: A multi-strategy proof-planner. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, July 2005. Springer.
14. J. D. C. Richardson, A. Smaill, and I. M. Green. System description: proof planning in higher-order logic with λ -clam. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-98)*, volume 1421 of LNAI. Springer, 1998.
15. V. Sorge. *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*. PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, November 2001.
16. B. Wack. *Typage et déduction dans le calcul de réécriture*. Thèse de doctorat, Université Henri Poincaré (Nancy 1), octobre 2005.
17. F. Wiedijk. Formal proof sketches. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs: Third International Workshop, TYPES 2003*, LNCS 3085, pages 378–393, Torino, Italy, 2004. Springer.