Combining Proofs of Higher-Order and First-Order Automated Theorem Provers

Christoph Benzmüller[1], Volker Sorge[2], Mateja Jamnik[3], and Manfred Kerber[2]

[1]Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany (www.ags.uni-sb.de/~chris)
[2]School of Computer Science, The University of Birmingham
Birmingham B15 2TT, England, UK (www.cs.bham.ac.uk/~vxs)
[3]University of Cambridge Computer Laboratory
Cambridge CB3 0FD, England, UK (www.cl.cam.ac.uk/~mj201)

Abstract. ΩANTS is an agent-oriented environment for combining inference systems. A characteristics of the ΩANTS approach is that a common proof object is generated by the cooperating systems. This common proof object can be inspected by verification tools to validate the correctness of the proof. ΩANTS makes use of a two layered blackboard architecture, in which the applicability of inference rules are checked on one (abstract) layer. The lower layer administrates explicit proof objects in a common language. In concrete proofs these proof objects can be quite bit, which can make communication during proof search very inefficient. As a result we had situations in which most of the resources went into the overhead of constructing explicit proof objects and communicating between different components. Therefore we have recently developed an alternative modelling of cooperating systems in ΩANTS which allows direct communication between related systems during proof search. This has the consequence that proof objects can no longer be directly constructed and thus the correctness-validation in this novel approach is in question. In this paper we present a pragmatic approach how this can rectified.

1 Introduction

ΩANTS is an agent-oriented environment for combining inference rules and inference systems. ΩANTS was originally conceived to support interactive theorem proving but was later extended to a fully automated proving system [23, 8]. A characteristics of the ΩANTS approach is that a joint proof object is generated by the cooperating inference rules and inference systems. This joint proof object can be inspected by proof verification tools in combination with proof expansion in order to validate the correctness at a purely logic level. The ΩANTS blackboard architecture consists of two layers, an abstract upper layer, and a more detailed lower layer. Applicability criteria for inference rules are modelled at the upper layer. The upper layer is supported by computations at the lower layer which models criteria for the instantiation of the parameters of the inference rules.

External systems have been modelled in ΩANTS as individual inference rules at the upper layer. With this approach, ΩANTS has been successfully employed in past experiments to check the validity of set equations using higher-order and first-order theorem provers, model generation, and computer algebra [5]. However, this approach was very inefficient for hard examples because of the communication overhead imposed by the need to translate all steps into a common proof data structure.

Therefore, we have recently developed an alternative approach: *the single inference rule approach* of cooperating systems in ΩANTS which exploits the lower layer of the blackboard architecture. This approach has been successfully applied to the combination of automated higher-order and first-order theorem provers. In particular, it has outperformed state-of-the-art first-order specialist reasoners (including Vampire 7.0) on 45 examples on sets, relations and functions; see [9].

Unfortunately, using a single inference rule approach, we had to sacrifice the generation of joint proof objects and correctness validation in this novel approach. In this paper we present a pragmatic approach to how this can be rectified.

The paper is structured as follows: In Section 2 we motivate and illustrate the cooperation between a higher-order theorem prover (we employ LEO [6]) and a first-order theorem prover (we employ BLIKSEM [12]). In Section 3 we compare the two options for modelling cooperative reasoning systems in ΩANTS: the initial multiple inference approach and the novel single inference rule approach. In Section 4 we show how a joint proof object can also be obtained for the latter modelling by mapping it back to the former. Section 5 concludes the paper.

2 Combining Higher-Order and First-Order ATP

2.1 Motivation

When dealing with problems containing higher-order concepts, such as sets, functions, or relations, today's state-of-the-art first-order automated theorem provers (ATPs) still exhibit weaknesses on problems considered relatively simple by humans (cf. [15]). One reason is that the problem formulations use an encoding in a first-order set theory, which makes it particularly challenging when trying to prove theorems from first principles, that is, basic axioms. Therefore, to aid ATPs in finding proofs, problems are often enriched by hand-picked additional lemmata, or axioms of the selected set theory are dropped leaving the theory incomplete. This has recently motivated extensions of state-of-the-art first-order calculi and systems, as for example presented in [15] for the SATURATE system. The extended SATURATE system can solve some problems from the SET domain in the TPTP [25] which VAMPIRE [22] and E-SETHEO's [24] cannot solve.

While it has already been shown in [6, 2] that many problems of this nature can be easily proved from first principles using a concise higher-order representation and the higher-order resolution ATP LEO [6], the combinatorial explosion inherent in LEO's calculus prevents the prover from solving a whole range of

SET171+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
SET611+3	$\forall X_{o\alpha}, Y_{o\alpha}.(X \cap Y = \emptyset) \Leftrightarrow (X \setminus Y = X)$
SET624+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.\text{Meets}(X, Y \cap Z) \Leftrightarrow \text{Meets}(X, Y) \vee \text{Meets}(X, Z)$
SET646+3	$\forall x_\alpha, y_\beta.\text{Subrel}(\text{Pair}(x, y), (\lambda u_\alpha.\top) \times (\lambda v_\beta.\top))$
SET670+3	$\forall Z_{o\alpha}, R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.\text{IsRelOn}(R, X, Y) \Rightarrow$ $\text{IsRelOn}(\text{RestrictRDom}(R, Z), Z, Y)$

Table 1. Test Problems on Sets and Relations: Examples

possible problems with one universal strategy. Often higher-order problems require only relatively few but essential steps of higher-order reasoning, while the overwhelming part of the reasoning is first-order or even propositional level. This suggests that LEO's performance could be improved when combining it with a first-order ATP to search efficiently for a possible refutation in the subset of those clauses that are essentially first-order.

The advantages of such a combination are not only that many problems can still be efficiently shown from first principles in a general purpose approach, but also that problems can be expressed in a very concise way.

For instance, in [9] we present 45 problems from the SET domain of the TPTP-v3.0.1, together with their entire formalisation in less than two pages, which is difficult to achieve within a framework that does not provide λ-abstraction. We have used this problem set, which is an extension of the problems considered in [15], to show the effectiveness of our approach (cf. [9]). While many of the considered problems can be proved by LEO alone with some strategy, the combination of LEO with the first-order ATP BLIKSEM [12] is not only able to solve more problems, but also needs only a single strategy to prove them. Several of our problems are considered very challenging by the first-order community and five of them (of which LEO can solve four) have a TPTP rating of 1.00, saying that they cannot be solved by any TPTP prover to date.

Technically, the combination has been realised in the concurrent reasoning system ΩANTS [23, 8] which enables the cooperation of hybrid reasoning systems to construct a common proof object. In our past experiments, ΩANTS has been successfully employed to check the validity of set equations using higher-order and first-order ATPs, model generation, and computer algebra [5]. While this enabled a cooperation between LEO and a first-order ATP, the proposed solution could not be classified as a general purpose approach. A major shortcoming was that all communication of partial results had to be conducted via the common proof object, which was very inefficient for hard examples. Thus, the solved examples from set theory were considered too trivial, albeit they were often similar to those still considered challenging in the TPTP in the first-order context.

In [9] we have presented our novel approach to the cooperation between LEO and BLIKSEM inside ΩANTS by decentralising communication. As has been documented in [9] this leads not only to a higher overall efficiency but also to a general purpose approach based on a single strategy in LEO.

2.2 Sets, Relations, and Functions: Higher-Order Logic Encoding

We list some examples of the test problems on sets and relations (and functions) that have been investigated in [9]. These test problems are taken from the TPTP against which we evaluated our system. The problems are given by the identifiers used in the SET domain of the TPTP, and are formalized in a variant of Church's simply typed λ-calculus with prefix polymorphism. In classical type theory, terms and all their sub-terms are typed. Polymorphism allows the introduction of type variables such that statements can be made for all types. For instance, in problem SET171 in Table 1, the universally quantified variable $X_{o\alpha}$ denotes a mapping from objects of type α to objects of type o. We use Church's notation $o\alpha$, which stands for the functional type $\alpha \to o$. The reader is referred to [1] for a more detailed introduction. In the remainder, o will denote the type of truth values, and small Greek letters will denote arbitrary types. Thus, $X_{o\alpha}$ (and its η-longform $\lambda y_{\alpha}.Xy$) is actually a characteristic function denoting the set of elements of type α, for which the predicate associated with X holds. As further notational convention, we use capital letter variables to denote sets, functions, or relations, while lower case letters denote individuals. Types are usually only given in the first occurrence of a variable and omitted if inferable from the context. Table 1 presents some examples of the test problems investigated in [9].

These test problems employ defined concepts that are specified in a knowledge base of hierarchical theories that LEO has access to. Table 2 gives the concepts necessary for defining the above problems:

$\_ \in \_$	$:= \lambda x_{\alpha}, A_{o\alpha}.[Ax]$
$\emptyset$	$:= [\lambda x_{\alpha}.\bot]$
$\_ \cap \_$	$:= \lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_{\alpha}.x \in A \wedge x \in B]$
$\_ \cup \_$	$:= \lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_{\alpha}.x \in A \vee x \in B]$
$\_ \setminus \_$	$:= \lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_{\alpha}.x \in A \vee x \notin B]$
Meets$(_,_)$	$:= \lambda A_{o\alpha}, B_{o\alpha}.[\exists x_{\alpha}.x \in A \wedge x \in B]$
Pair$(_,_)$	$:= \lambda x_{\alpha}, y_{\beta}.[\lambda u_{\alpha}, v_{\beta}.u = x \wedge v = y]$
$\_ \times \_$	$:= \lambda A_{o\alpha}, B_{o\beta}.[\lambda u_{\alpha}, v_{\beta}.u \in A \wedge v \in B]$
Subrel$(_,_)$	$:= \lambda R_{o\beta\alpha}, Q_{o\beta\alpha}.[\forall x_{\alpha}, y_{\beta}.Rxy \Rightarrow Qxy]$
IsRelOn$(_,_,_)$	$:= \lambda R_{o\beta\alpha}, A_{o\alpha}, B_{o\beta}.[\forall x_{\alpha}, y_{\beta}.Rxy \Rightarrow x \in A \wedge y \in B]$
RestrictRDom$(_,_)$	$:= \lambda R_{o\beta\alpha}, A_{o\alpha}.[\lambda x_{\alpha}, y_{\beta}.x \in A \wedge Rxy]$

Table 2. Definitions of Operations on Sets and Relations: Examples

These concepts are defined in terms of λ-expressions and they may contain other, already specified concepts. For presentation purposes, we use customary mathematical symbols $\cup, \cap$, etc., for some concepts like *union, intersection*, etc., and we also use infix notation. For instance, the definition of union on sets in Table 2 can be easily read in its more common mathematical representation $A \cup B := \{x | x \in A \vee x \in B\}$. Before proving a problem, LEO always expands —

Assumptions:	$\forall B, C, x_{\blacksquare}[x \in (B \cup C) \Leftrightarrow x \in B \vee x \in C]$	(1)
	$\forall B, C, x_{\blacksquare}[x \in (B \cap C) \Leftrightarrow x \in B \wedge x \in C]$	(2)
	$\forall B, C_{\blacksquare}[B = C \Leftrightarrow B \subseteq C \wedge C \subseteq B]$	(3)
	$\forall B, C_{\blacksquare}[B \cup C = C \cup B]$	(4)
	$\forall B, C_{\blacksquare}[B \cap C = C \cap B]$	(5)
	$\forall B, C_{\blacksquare}[B \subseteq C \Leftrightarrow \forall x_{\blacksquare} x \in B \Rightarrow x \in C]$	(6)
	$\forall B, C_{\blacksquare}[B = C \Leftrightarrow \forall x_{\blacksquare} x \in B \Leftrightarrow x \in C]$	(7)
Proof Goal:	$\forall B, C, D_{\blacksquare}[B \cup (C \cap D) = (B \cup C) \cap (B \cup D)]$	(8)

Table 3. Problem SET171+3: The First-Order TPTP Encoding.

recursively, if necessary — all occurring concepts. This straightforward expansion to first principles is realised by an automated preprocess in our current approach.

2.3 Sets, Relations, and Functions: First-Order Logic Encoding

Let us consider example SET171+3 in its first-order formulation from the TPTP (see Table 3). We can observe that the assumptions provide only a partial axiomatisation of naive set theory. On the other hand, the specification introduces lemmata that are useful for solving the problem. In particular, assumption (7) is trivially derivable from (3) with (6). Obviously, clausal normalisation of this first-order problem description yields a much larger and more difficult set of clauses. Furthermore, definitions of concepts are not directly expanded as in Leo. It is therefore not surprising that most first-order ATPs still fail to prove this problem. In fact, very few TPTP provers were successful in proving SET171+3. Amongst them are Muscadet 2.4. [21], Vampire 7.0, and Saturate. The natural deduction system Muscadet uses special inference rules for sets and needs 0.2 seconds to prove this problem. Vampire needs 108 seconds. The Saturate system [15] (which extends Vampire with Boolean extensionality rules that are a one-to-one correspondence to Leo's rules for Extensional Higher-Order Paramodulation [3]) can solve the problem in 2.9 seconds while generating 159 clauses. The significance of such comparisons is clearly limited since different systems are optimised to a different degree. One noted difference between the experiments with first-order provers listed above, and the experiments with Leo and Leo-Bliksem is that first-order systems often use a case tailored problem representation (e.g., by avoiding some base axioms of the addressed theory), while Leo and Leo-Bliksem have a harder task of dealing with a general (not specifically tailored) representation. Thus, the comparison of the performance of Leo and Leo-Bliksem with first-order systems as done in [9] is unfair: the higher-order systems attack harder, non-tailored problems. Nevertheless, as we demonstrated by the performance results in [9] the higher-order systems still perform better.

(1) $\forall B, C, D.B \cup (C \cap D) = (B \cup C) \cap (B \cup D)$	
	↓ clause initialization
	↓ def.-expansion, cnf
	↓ B, C, D Skolem const.
(2) $[(\lambda x.Bx \vee (Cx \wedge Dx)) = (\lambda x.(Bx \wedge Cx) \vee (Cx \wedge Dx))]^F$	
	↓ unification constraint
(3) $[(\lambda x.Bx \vee (Cx \wedge Dx)) =^? (\lambda x.(Bx \wedge Cx) \vee (Cx \wedge Dx))]$	
	↓ f-extensionality
	↓ x new Skolem constant
(4) $[(Bx \vee (Cx \wedge Dx)) =^? ((Bx \wedge Cx) \vee (Cx \wedge Dx))]$	
	↓ B-extensionality
(5) $[(Bx \vee (Cx \wedge Dx)) \Leftrightarrow ((Bx \wedge Cx) \vee (Cx \wedge Dx))]^F$	
	↓ cnf, factor., subsumption
(6) $[Bx]^F$	
(7) $[Bx]^T \vee [Cx]^T$	propositional problem!
(8) $[Bx]^T \vee [Dx]^T$	
(9) $[Cx]^F \vee [Dx]^F$	
	↓ propositional reasoning
(10) $\square$	

Table 4. Problem SET171+3: Solution in LEO

For the experiments with LEO and the cooperation of LEO with the first-order theorem prover BLIKSEM, λ-abstraction as well as the extensionality treatment inherent in LEO's calculus [4] is used. This enables a theoretically[1] Henkin-complete proof system for set theory. In the above example SET171+3, LEO generally uses the application of functional extensionality to push extensional unification constraints down to base type level, and then eventually applies Boolean extensionality to generate clauses from them. These are typically much simpler and often even *propositional-like* or *first-order-like* (FO-like, for short), that is, they do not contain any 'real' higher-order subterms (such as a λ-abstraction or embedded equations), and are therefore suitable for treatment by a first-order ATP or even a propositional logic decision procedure.

2.4 Solving the Test Problem SET171+3 in LEO

Table 4 illustrates how LEO tackles and solves the test problem SET171+3. First the resolution process is initialised, that is, the definitions occurring in the input problem are expanded, that is, completely reduced to first principles. Then the problem is turned into a negated unit clause. The resulting (not displayed intermediate) clause is not in normal form and therefore LEO first normalizes it with explicit clause normalisation rules (cnf) to reach some proper initial clauses. In

[1] For pragmatic reasons, such as efficiency, most of LEO's tactics are incomplete. LEO's philosophy is to rely on a theoretically complete calculus, but to practically provide a set of complimentary strategies so that these cover a broad range of theorems.

our concrete case, this leads to the unit clause (2). Note that negated primitive equations are generally automatically converted by LEO into unification constraints. This is why (2) is automatically converted into (3), which is a syntactically not solvable, but is a semantic unification problem. Observe, that we write $[.]^T$ and $[.]^F$ for positive and negative literals, respectively. LEO then applies its goal directed functional and Boolean extensionality rules which replace the unification constraint (3) by the clauses (4) and (5). Unit clause (5) is again not normal; normalisation, factorisation and subsumption yields the clauses (6)-(9). This set is essentially of propositional logic character and trivially refutable. LEO needs 0.56 seconds for solving the problem and generates a total of 36 clauses.

2.5 Solving the Test Problem SET171+3 in LEO-BLIKSEM

As illustrated in Table 4, LEO transforms test problem SET171+3 straightforwardly into a propositional like subproblem. Here the generated clause set (7)–(10) can still be efficiently refuted by LEO. Generally, however, the generated subsets of propositional or first-order like subproblems may quickly become so big that LEO's refutation procedure, which is not optimised for these problem classes, gets stuck. And in LEO's search space generally some further real higher-order clauses have to be taken into account. This observation motivates our cooperative LEO-BLIKSEM proof search approach: while LEO performs its proof search as before, it periodically also passes the detected first-order like clauses (which, of course, include the propositional like clauses) to the first-order specialist reasoner BLIKSEM. We note:

- The generated first-order like clauses in LEO are copied into a special bag which never decreases and usually always increases. That is, the bag of first-order like clauses dynamically changes and eventually becomes refutable (such as clauses (7)–(10) in our example).
- LEO's proof search procedure remains unchanged and LEO still tries to refute such subproblems itself (as before) in a bigger context.
- In addition, specialist reasoners may now support LEO by showing that the bag of first-order like subproblems is refutable.
- Each time the bag of first-order like subproblems is increased by LEO, a new instance of a specialist reasoner is launched (with a resource-bound). This instance runs in parallel to LEO's proof search and may eventually signal success to LEO. If LEO receives such a success signal, it stops its own proof search and reports that a cooperative proof has been found. Alternatively (as before) LEO stops proof search when it finds the proof itself.
- Our cooperative approach can easily be fine-grained by separating the bag of first-order like clauses into even more specialised subclasses, such as propositional logic, guarded fragment, etc. Different specialist reasoners can then be employed to attack these clause sets.
- For the higher-order problems investigated in [9] we further observe:
  - Some problems are immediately mapped by recursive definition expansion (without extensionality reasoning) and normalisation into first-order like problems; an example is SET624+3.

- Some problems are immediately mapped by recursive definition expansion (without extensionality reasoning) and normalisation into the empty clause such that proof search does not even start; an example is SET646+3.
- Some problems require several rounds of extensionality processing within LEO's set-of-support based proof search procedure before the bag of first-order like clauses turns into a refutable set of clauses; an example is SET611+3.

The result of the case study performed in [9] is: The LEO-BLIKSEM cooperation impressively outperforms both state-of-the art first-order specialists (including Vampire 7.0) and the non-cooperative LEO system.

In the next section we describe in more detail how the cooperative proof search approach between LEO and the first-order prover BLIKSEM has been modelled in ΩANTS.

3 ΩANTS

ΩANTS was originally conceived to support interactive theorem proving but was later extended to a fully automated proving system [23, 8]. Its basic idea is to compose a *central proof object* by generating, in each proof situation, a ranked list of potentially applicable inference steps. In this process, all inference rules, such as calculus rules or tactics, are uniformly viewed with respect to three sets: premises, conclusions, and additional parameters. The elements of these three sets are called *arguments* of the inference rule and they usually depend on each other. An inference rule is applicable if at least some of its arguments can be instantiated with respect to the given proof context. The task of the ΩANTS architecture is now to determine the applicability of inference rules by computing instantiations for their arguments.

The architecture consists of two layers. On the lower layer, possible instantiations of the arguments of individual inference rules are computed. In particular, each inference rule is associated with its own blackboard and concurrent processes, one for each argument of the inference rule. The role of every process is to compute possible instantiations for its designated argument of the inference rule, and to record these on the blackboard. The computations are carried out with respect to the given proof context and by exploiting information already present on the blackboard, that is, argument instantiations computed by other processes. On the upper layer, the information from the lower layer is used for computing and heuristically ranking the inference rules that are applicable in the current proof state. The most promising rule is then applied to the central proof object and the data on the blackboards is cleared for the next round of computations.

ΩANTS employs resource reasoning to guide search.[2] This enables the controlled integration (e.g., by specifying time-outs) of full-fledged external reason-

[2] ΩANTS provides facilities to define and modify the processes at run-time. But notice that we do not use these advanced features in the case study presented in this paper.

ing systems such as automated theorem provers, computer algebra systems, or model generators into the architecture.

3.1 Cooperation via multiple inference rules

The use of the external systems is modelled by inference rules, usually one for each system. Their corresponding computations are encapsulated in one of the independent processes in the architecture. For example, an inference rule modelling the standard application LEO has its conclusion argument set to be an open higher-order (HO) goal.

$$\frac{}{\textbf{HO-goal}}\ \textsc{Leo}\ \textbf{(LEO-parameters)}$$

A process can then place an open goal on the blackboard, where it is picked up by a process that applies the LEO prover to it. Any computed proof from the external system is again written to the blackboard from where it is subsequently inserted into the proof object when the inference rule is applied. While this setup enables proof construction by a collaborative effort of diverse reasoning systems, the cooperation can only be achieved via the central proof object. This means that all partial results have to be translated back and forth between the syntaxes of the integrated systems and the language of the proof object. For modelling the cooperation of LEO with a first-order reasoner we have first experimented with the following multiple inference rule modelling (see also [5]):

$$\frac{\textbf{Neg-Conj-of-FO-clauses}}{\textbf{HO-goal}}\ \textsc{Leo}\text{-with-partial-result}\textbf{(Leo-parameters)}$$

$$\frac{}{\textbf{FO-goal}}\ \textsc{Bliksem}\ \textbf{(Bliksem-parameters)}$$

The first rule models a process that picks up higher-order proof problem from the blackboard, passes it to LEO which starts its proof search, and then returns the negated conjunction of generated first-order clauses back (e.g. the negated conjunction of the clauses (7)–(10) in our previous example). For each modified bag of first-order like clauses in LEO this rule may suggest a novel reduction of the original higher-order goal to a first-order criterion.

Since there are many types of integrated systems, the language of the proof object maintained in ΩANTS — a higher-order language even richer than LEO's, together with a natural deduction calculus — is expressive but also cumbersome. This leads not only to a large communication overhead, but also means that complex proof objects have to be created, even if the reasoning of all systems involved is clause-based. Large clause sets need to be transformed into large single formulae to represent them in the proof object; the support for this in ΩANTS to date is inefficient. Consequently, the cooperation between external systems is typically rather inefficient [5].

3.2 Cooperation via a single inference rule

In order to overcome the problem of the communication bottleneck described above, we devised a new method for the cooperation between a higher-order and a first-order theorem prover within ΩANTS. Rather than modelling each theorem prover as a separate inference rule (and hence needing to translate the communication via the language of the central proof object), we model the cooperation between a higher-order (concretely, LEO) and a first-order theorem prover (in our case study BLIKSEM) in ΩANTS as a single inference rule.

$$\frac{}{\textbf{HO-goal}}\ \text{LEO-BLIKSEM}\ (\textbf{Leo-partial-proof, FO-clauses, FO-proof, Leo-parameters, Bliksem-parameters})$$

The communication between the two theorem provers is carried out directly by the parameters of the inference rule and not via the central proof object. This avoids translating clause sets into single formulae and back.

Concretely, the single inference rule modelling the cooperation between LEO and a first-order theorem prover needs the following arguments to be applicable: (1) an open higher-order proof goal, (2) a partial LEO proof, (3) a set of FO-like clauses in the partial proof, (4) a first-order refutation proof for the set of FO-like clauses, and (5) and (6) the usual flag-parameters for the theorem provers LEO and BLIKSEM. Each of these arguments is computed, that is, its instantiation is found, by an independent process. The first process finds open goals in the central proof object and posts them on the blackboard associated with the new rule. The second process starts an instance of the LEO theorem prover for each new open goal on the blackboard. Each LEO instance maintains its own set of FO-like clauses. The third process monitors these clauses, and as soon as it detects a change in this set, that is, if new FO-like clauses are added by LEO, it writes the entire set of clauses to the blackboard. Once FO-like clauses are posted, the fourth process first translates each of the clauses directly into a corresponding one in the format of the first-order theorem prover, and then starts the first-order theorem prover on them. Note that writing FO-like clauses on the blackboard is by far not as time consuming as generating higher-order proof objects. As soon as either LEO or the first-order prover finds a refutation, the second process reports LEO's proof or partial proof to the blackboard, that is, it instantiates argument (2). Once all four arguments of our inference rule are instantiated, the rule becomes applicable and its application closes the open proof goal in the central proof object. That is, the open goal is proved by the cooperation between LEO and a first-order theorem prover. When computing applicability of the inference rule, the second and the fourth process concurrently spawn processes running LEO or a first-order prover on a different set of FO-like clauses. Thus, when actually applying the inference rule, all these instances of provers working on the same open subgoal are stopped.

While in the previous approach with multiple inference rules the cooperation between LEO and BLIKSEM was modelled at the upper layer of the ΩANTS architecture, our new approach models their cooperation by exploiting the lower

layer of the ΩANTS blackboard architecture. This is not an ad hoc solution, but rather, it demonstrates ΩANTS's flexibility in modelling the integration of cooperative reasoning systems.

Our approach to the cooperation between a higher-order and a first-order theorem prover has many advantages. The main one is that the communication is restricted to the transmission of clauses, and thus it avoids any intermediate translation into the language of the central proof object. This significantly reduces the communication overhead and makes effective proving of more involved theorems feasible.

4 Constructing a Combined Proof Object

A disadvantage of our approach is that we cannot easily translate and integrate the two proof objects produced by LEO and BLIKSEM into the central proof object maintained by ΩANTS. This has been possible in our previous approach with multiple inference rules. Thus, we developed a simple and pragmatic solution to the problem:

- The main idea is to replay the proof on the upper level of the ΩANTS architecture (using the multiple inference rule modelling) once a proof attempt was successful (with a single inference rule modelling) on the lower level.
- We can essentially reconstruct all the information from the blackboard that we need in order to replay the proof. For this remember that the rule LEO-BLIKSEM is only applicable if all parameters of the rule are instantiated, that is, the respective parameter instantiation information is available on the blackboard for each successful cooperative proof attempt. Respective instantiation information generated from a successful cooperative proof attempt for our running example SET171+3, for instance, is:

$$
\begin{aligned}
\textbf{HO-Goal} &:= \forall B, C, D.C \cup (B \cap D) = (C \cup B) \cap (C \cup D) \\
\textbf{Leo-partial-proof} &:= \ldots \textit{ a HO resolution proof object } \Delta \ \ldots \\
\textbf{FO-clauses} &:= (7) \ \ [Bx]^F \\
&\qquad\quad (8) \ \ [Bx]^T \vee [Cx]^T \\
&\qquad\quad (9) \ \ [Bx]^T \vee [Dx]^T \\
&\qquad\quad (10) \ \ [Cx]^F \vee [Dx]^F \\
\textbf{FO-proof} &:= \ldots \textit{ a FO resolution proof object } \Gamma \ \ldots \\
\textbf{Leo-parameters} &:= \ldots \textit{ the flags chosen for the } \text{LEO} \textit{ call} \ldots \\
\textbf{Bliksem-parameters} &:= \ldots \textit{ the flags chosen for the } \text{BLIKSEM} \textit{ call} \ldots
\end{aligned}
$$

- For finding joint proofs efficiently in our experiment we called BLIKSEM in the fastest mode. In this case the generated FO-proof object is typically very sparse, i.e. contains only very little information for proof reconstruction and transformation.

– When the above suggestion of a successful joint proof attempt is selected for application in ΩANTS, the initially open (sub-)goal $\forall B, C, D. C \cup (B \cap D) = (C \cup B) \cap (C \cup D)$ is closed and the new justification of this proof node becomes 'LEO-BLIKSEM' augmented with the above parameter instantiation information:

$$\frac{}{\forall B, C, D. C \cup (B \cap D) = (C \cup B) \cap (C \cup D)}\ \text{LEO-BLIKSEM}\ \textit{(above param. inst.)}$$

– Expansion of this node then replaces the (sub-)proof object by the following (sub-)proof object employing the multiple inference rule modelling of the cooperative proof attempt:

$$\dfrac{\dfrac{}{\textit{neg-FO-clauses}}\ \text{BLIKSEM}\ \textit{(modified Bliksem-param. instantiation)}}{\forall B, C, D.\ \ldots = \ldots}\ \text{LEO-}\textit{with-partial-result (Leo-param. instantiation)}$$

where '*neg-FO-clauses*' is computed from the instantiation of the parameter **FO-clauses** as

$$\neg(\neg(Bx) \wedge (Bx \vee Cx) \wedge (Bx \vee Dx) \wedge (\neg(Cx) \vee \neg(Dx))$$

– The idea is to support verification of this (sub-)proof by subsequent proof node expansion, i.e., to investigate the contributions of both reasoning systems separately. For the expansion of BLIKSEM, a translation of the previously generated proof into a proper proof-object is not an option if we called BLIKSEM in the fastest mode since the delivered first-order proof object may be too sparse. Therefore, the expansion of this proof node simply calls BLIKSEM again but now within a different mode (determined by the slightly changed *modified Bliksem-param. instantiation*) which ensures the generation of detailed first-order proof objects.
– For the translation of this regenerated, detailed first-order proof object into an ΩANTS proof object we employ the TRAMP system [18]. This enables us to verify the (sub-)proof of BLIKSEM after its translation into an ΩANTS proof object.
– Generally, we could also replace the second call to BLIKSEM by a call to any other first-order proof system that is supported by TRAMP's generic proof transformation mechanism (and which is as strong as BLIKSEM).

5 Conclusion

In this paper we have discussed the difference between two forms of modelling cooperating proof systems within ΩANTS: the multiple inference rule approach and the single inference rule approach. In previous experiments the latter has been shown as highly efficient and it has outperformed state-of-the-art first-order specialist reasoners on 45 examples on sets, relations and functions; cf. [9]. The drawback so far, however, was that no joint proof object could be generated. In

this paper we have reported how we have solved this problem by simply mapping the single inference rule modelling back to the multiple inference rule modelling.

Related to our approach is the TECHS system [13], which realises a cooperation between a set of heterogeneous first-order theorem provers. Similarly to our approach, partial results in TECHS are exchanged between the different theorem provers in form of clauses. The main difference to the work of Denzinger *et al.* (and other related architectures like [14]) is that our system bridges between higher-order and first-order automated theorem proving. Also, unlike in TECHS, we provide a declarative specification framework for modelling external systems as cooperating, concurrent processes that can be (re-)configured at run-time. Related is also the work of Hurd [16] which realises a generic interface between HOL and first-order theorem provers. It is similar to the solution previously achieved by TRAMP [18] in OMEGA, which serves as a basis for the sound integration of ATPs into ΩANTS. Both approaches pass essentially first-order clauses to first-order theorem provers and then translate their results back into HOL resp. OMEGA. Some further related work on the cooperation of Isabelle with VAMPIRE is presented in [19]. The main difference of our work to the related systems is that while our system calls first-order provers from within higher-order proof search, this is not the case for [16, 18, 19].

Future work is to investigate how far our approach scales up to more complex problems and more advanced mathematical theories. In less trivial settings as discussed in this paper, we will face the problem of selecting and adding relevant lemmata to avoid immediate reduction to first principles and to appropriately instantiate set variables. Relevant related work for this setting is Bishop's approach to *selectively expand definitions* as presented in [10] and Brown's PhD thesis on *set comprehension in Church's type theory* [11].

References

1. P. Andrews. *An Introduction to mathematical logic and Type Theory: To Truth through Proof.* Number 27 in Applied Logic Series. Kluwer, 2002.
2. C. Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving.* PhD thesis, Universität des Saarlandes, Germany, 1999.
3. C. Benzmüller. Extensional higher-order paramodulation and RUE-resolution. *Proc. of CADE-16*, *LNAI* 1632, p. 399–413. Springer, 1999.
4. C. Benzmüller. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–235, 2002.
5. C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. Experiments with an Agent-Oriented Reasoning System. *Proc. of KI 2001*, *LNAI* 2174, p.409–424. Springer, 2001.
6. C. Benzmüller and M. Kohlhase. LEO – a higher-order theorem prover. *Proc. of CADE-15*, *LNAI* 1421. Springer, 1998.
7. C. Benzmüller and V. Sorge. A Blackboard Architecture for Guiding Interactive Proofs. *Proc. of AIMSA'98*, *LNAI* 1480, p. 102–114. Springer, 1998.
8. C. Benzmüller and V. Sorge. ΩANTS – An open approach at combining Interactive and Automated Theorem Proving. *Proc. of Calculemus-2000.* AK Peters, 2001.
9. C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Can a Higher-Order and a First-Order Theorem Prover Cooperate? Proc. LPAR'04, LNAI 3452, Montevideo, Uruguay. Springer, 2005.

10. M. Bishop and P. Andrews. Selectively instantiating definitions. *Proc. of CADE-15*, *LNAI* 1421. Springer, 1998.
11. C. E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Dept. of Mathematical Sciences, Carnegie Mellon University, USA, 2004.
12. H. de Nivelle. *The Bliksem Theorem Prover, Version 1.12*. Max-Planck-Institut, Saarbrücken, Germany, 1999. http://www.mpi-sb.mpg.de/ bliksem/manual.ps.
13. J. Denzinger and D. Fuchs. Cooperation of Heterogeneous Provers. *Proc. IJCAI-16*, p. 10–15. Morgan Kaufmann, 1999.
14. M. Fisher and A. Ireland. Multi-agent proof-planning. CADE-15 Workshop "Using AI methods in Deduction", 1998.
15. H. Ganzinger and J. Stuber. Superposition with equivalence reasoning and delayed clause normal form transformation. *Proc. of CADE-19*, *LNAI* 2741. Springer, 2003.
16. J. Hurd. An LCF-style interface between HOL and first-order logic. *Automated Deduction — CADE-18*, *LNAI* 2392, p. 134–138. Springer, 2002.
17. M. Kerber. *On the Representation of Mathematical Concepts and their Translation into First Order Logic*. PhD thesis, Universität Kaiserslautern, Germany, 1992.
18. A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. *Proc. of CADE-17*, *LNAI* 1831. Springer, 2000.
19. J. Meng and L. C. Paulson. Experiments on supporting interactive proof using resolution. *Proc. of IJCAR 2004*, *LNCS* 3097, p. 372–384. Springer, 2004.
20. R. Nieuwenhuis, Th. Hillenbrand, A. Riazanov, and A. Voronkov. On the evaluation of indexing techniques for theorem proving. *Proc. of IJCAR-01*, *LNAI* 2083, p. 257–271. Springer, 2001.
21. D. Pastre. Muscadet2.3 : A knowledge-based theorem prover based on natural deduction. *Proc. of IJCAR-01*, *LNAI* 2083, p. 685–689. Springer, 2001.
22. A. Riazanov and A. Voronkov. Vampire 1.1 (system description). *Proc. of IJCAR-01*, *LNAI* 2083, p. 376–380. Springer, 2001.
23. V. Sorge. *OANTS: A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*. PhD thesis, Universität des Saarlandes, Germany, 2001.
24. G. Stenz and A. Wolf. E-SETHEO: An Automated^3 Theorem Prover – System Abstract. *Proc. of the TABLEAUX'2000*, *LNAI* 1847, p. 436–440. Springer, 2000.
25. G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.