Can a Higher-Order and a First-Order Theorem Prover Cooperate?*

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|mmk})
[3] University of Cambridge Computer Laboratory
Cambridge CB3 0FD, England, UK (www.cl.cam.ac.uk/~mj201)

Abstract. State-of-the-art first-order automated theorem proving systems have reached considerable strength over recent years. However, in many areas of mathematics they are still a long way from reliably proving theorems that would be considered relatively simple by humans. For example, when reasoning about sets, relations, or functions, first-order systems still exhibit serious weaknesses. While it has been shown in the past that higher-order reasoning systems can solve problems of this kind automatically, the complexity inherent in their calculi and their inefficiency in dealing with large numbers of clauses prevent these systems from solving a whole range of problems.
We present a solution to this challenge by combining a higher-order and a first-order automated theorem prover, both based on the resolution principle, in a flexible and distributed environment. By this we can exploit concise problem formulations without forgoing efficient reasoning on first-order subproblems. We demonstrate the effectiveness of our approach on a set of problems still considered non-trivial for many first-order theorem provers.

1 Introduction

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. [14]). 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

* This work was supported by EPSRC grant GR/M22031 and DFG-SFB 378 (first author), EU Marie-Curie-Fellowship HPMF-CT-2002-01701 (second author), and EPSRC Advanced Research Fellowship GR/R76783 (third author).

F. Baader and A. Voronkov (Eds.): LPAR 2004, LNAI 3452, pp. 415–431, 2005.

calculi and systems, as for example presented in [14] for the SATURATE system. The extended SATURATE system can solve some problems from the SET domain in the TPTP [24] which VAMPIRE [21] and E-SETHEO's [23] 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, the combinatorial explosion inherent in LEO's calculus prevents the prover from solving a whole range of 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 — further discussed in Sec. 2 — 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, we present 45 problems from the SET domain of the TPTP-v3.0.1, together with their entire formalisation in less than two pages in this paper, which is difficult to achieve within a framework that does not provide λ-abstraction. We use this problem set, which is an extension of the problems considered in [14], in Sec. 4 to show the effectiveness of our approach. 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 [11] is not only able to show more problems, but also needs only a single strategy to solve 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 — described in more detail in Sec. 3 — has been realised in the concurrent reasoning system OANTS [22,8] which enables the cooperation of hybrid reasoning systems to construct a common proof object. In our past experiments, OANTS 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 already 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 this paper we now present a novel approach to the cooperation between LEO and BLIKSEM inside OANTS by decentralising communication. This leads not only to a higher overall efficiency — Sec. 4 details our results — but also to a general purpose approach based on a single strategy in LEO.

2 Why Linking Higher-Order and First-Order?

Existing higher-order ATPs generally exhibit deficits in efficiently reasoning with first-order problems for several reasons. Unlike in the case of first-order provers, for which sophisticated calculi and strategies, as well as advanced implementation techniques, such as term indexing [19], have been developed, fully mechanisable higher-order calculi are still at a comparably early stage of development. Some problems are much harder in higher-order, for instance, unification is undecidable, strong constraining term- and literal-orderings are not available, extensionality reasoning and set variable instantiation has to be addressed. Nevertheless, for some mathematical problem domains, such as naive set theory, for instance, automated higher-order reasoning performs very well.

We motivate the need for linking higher-order and first-order ATPs with some examples from Table 1. It contains a range of challenging problems taken from the TPTP, against which we will evaluate our system in Sec. 4. The problems are given by the identifiers used in the SET domain of the TPTP, and are formalised 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 SET014+4 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, $X_{o\alpha}$ (resp. 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.

The problems in Table 1 employ defined concepts that are specified in a knowledge base of hierarchical theories that Leo has access to. All concepts necessary for defining our problems in Table 1 are given in Table 2. 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 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 — recursively, if necessary — all occurring concepts. This straightforward expansion to first principles is realised by an automated preprocess in our current approach.

SET171+3 We first discuss example SET171+3 to contrast our formalisation to a standard first-order one. After recursively expanding the input problem, that is, completely reducing it to first principles, Leo turns it into a negated unit clause. Since this initial clause is not in normal form, Leo first normalises it with explicit

Table 1. Problems from TPTP for the evaluation of Oants

SET	Problem Formalisation
014+4	$\forall X_{o\alpha}, Y_{o\alpha}, A_{o\alpha}.[[X \subseteq A \wedge Y \subseteq A] \Rightarrow (X \cup Y) \subseteq A]$
017+1	$\forall x_\alpha, y_\alpha, z_\alpha.[UnOrderedPair(x,y) = UnOrderedPair(x,z) \Rightarrow y = z]$
066+1	$\forall x_\alpha, y_\alpha.[UnOrderedPair(x,y) = UnOrderedPair(y,x)$
067+1	$\forall x_\alpha, y_\alpha.[UnOrderedPair(x,x) \subseteq UnOrderedPair(x,y)]$
076+1	$\forall x_\alpha, y_\alpha.\forall Z_{o\alpha}.x \in Z \wedge y \in Z \Rightarrow UnOrderedPair(x,y) \subseteq Z$
086+1	$\forall x_\alpha.\exists y_\alpha.[y \in Singleton(x)]$
096+1	$\forall X_{o\alpha}, y_\alpha.[X \subseteq Singleton(y) \Rightarrow [X = \emptyset \vee X = Singleton(y)]]$
143+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[(X \cap Y) \cap Z = X \cap (Y \cap Z)]$
171+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)]$
580+3	$\forall X_{o\alpha}, Y_{o\alpha}, u_\alpha.[u \in ExclUnion(X,Y) \Leftrightarrow [u \in X \Leftrightarrow u \notin Y]]$
601+3	$\forall_. X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}[(X \cap Y) \cup ((Y \cap Z) \cup (Z \cap X)) = (X \cup Y) \cap ((Y \cup Z) \cap (Z \cup X))]$
606+3	$\forall X_{o\alpha}, Y_{o\alpha}.[X \backslash (X \cap Y) = X \backslash Y]$
607+3	$\forall X_{o\alpha}, Y_{o\alpha}.[X \cup (Y \backslash X) = X \cup Y]$
609+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[X \backslash (Y \backslash Z) = (X \backslash Y) \cup (X \cap Z)]$
611+3	$\forall X_{o\alpha}, Y_{o\alpha}.[X \cap Y = \emptyset \Leftrightarrow X \backslash Y = X]$
612+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[X \backslash (Y \cup Z) = (X \backslash Y) \cap (X \backslash Z)]$
614+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[(X \backslash Y) \backslash Z = X \backslash (Y \cup Z)]$
615+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[(X \cup Y) \backslash Z = (X \backslash Z) \cup (Y \backslash Z)]$
623+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[ExclUnion(ExclUnion(X,Y),Z) = ExclUnion(X, ExclUnion(Y,Z))]$
624+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[Meets(X,(Y \cup Z)) \Leftrightarrow [Meets(X,Y) \vee Meets(X,Z)]]$
630+3	$\forall X_{o\alpha}, Y_{o\alpha}.[Misses(X \cap Y, ExclUnion(X,Y))]$
640+3	$\forall R_{o\beta\alpha}, Q_{o\beta\alpha}.[Subrel(R,Q) \Rightarrow Subrel(R, (\lambda u_\alpha.\top) \times (\lambda v_\beta.\top))]$
646+3	$\forall x_\alpha, y_\beta.[Subrel(Pair(x,y), (\lambda u_\alpha.\top) \times (\lambda v_\beta.\top))]$
647+3	$\forall R_{o\beta\alpha}, X_{o\alpha}.[(RDom(R) \subseteq X) \Rightarrow Subrel(R, X \times RCodom(R))]$
648+3	$\forall R_{o\beta\alpha}, Y_{o\beta}.[(RCodom(R) \subseteq Y) \Rightarrow Subrel(R, RDom(R) \times Y)]$
649+3	$\forall R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[[RDom(R) \subseteq X \wedge RCodom(R) \subseteq Y] \Rightarrow Subrel(R, X \times Y)]$
651+3	$\forall R_{o\beta\alpha}.[RDom(R) \subseteq A_{o\alpha} \Rightarrow Subrel(R, A \times (\lambda u_\beta.\top))]$
657+3	$\forall R_{o\beta\alpha}.[Field(R) \subseteq ((\lambda u_\alpha.\top) \cup (\lambda v_\beta.\top))]$
669+3	$\forall R_{o\alpha\alpha}.[Subrel(Id(\lambda u_\alpha.\top), R) \Rightarrow [(\lambda u_\alpha.\top) \subseteq RDom(R) \wedge (\lambda u_\alpha.\top) = RCodom(R)]]$
670+3	$\forall Z_{o\alpha}, R_{o\beta\alpha}, X_{o\alpha} Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow IsRelOn(RestrictRDom(R,Z), Z, Y)]$
671+3	$\forall Z_{o\alpha}, R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[[IsRelOn(R,X,Y) \wedge X \subseteq Z] \Rightarrow RestrictRDom(R,Z) = R]$
672+3	$\forall Z_{o\beta}, R_{o\beta\alpha}, X_{o\alpha} Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow IsRelOn(RestrictRCodom(R,Z), X, Z)]$
673+3	$\forall Z_{o\beta}, R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[[IsRelOn(R,X,Y) \wedge Y \subseteq Z] \Rightarrow RestrictRCodom(R,Z) = R]$
680+3	$\forall R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow [\forall u_\alpha.u \in X \Rightarrow [u \in RDom(R) \Leftrightarrow \exists v_\beta.v \in Y \wedge R(u,v)]]]$
683+3	$\forall R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow [\forall v_\beta.v \in Y \Rightarrow [v \in RCodom(R) \Rightarrow \exists u_\alpha.u \in X \wedge u \in RDom(R)]]]$
684+3	$\forall P_{o\beta\alpha}, R_{o\gamma\beta}, x_\alpha, z_\gamma.[RelComp(P,R)xz \Leftrightarrow \exists y_\beta.Pxy \wedge Ryz]$
686+3	$\forall Z_{o\alpha}, R_{o\gamma\beta}, x_\alpha.[x \in InverseImageR(R,Z) \Leftrightarrow \exists y_\alpha.Rxy \wedge x \in Z]$
716+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}.[[Inj(F) \wedge Inj(G)] \Rightarrow Inj(G \circ F)]$
724+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}, H_{\gamma\beta}.[[F \circ G = F \circ H \wedge Surj(F)] \Rightarrow G = H]$
741+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}, H_{\alpha\gamma}.[[Inj((F \circ G) \circ H) \wedge Surj((G \circ H) \circ F) \wedge Surj((H \circ F) \circ G)] \Rightarrow Bij(H)]$
747+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}, \lhd^1_{o\alpha\alpha}, \lhd^2_{o\beta\beta}, \lhd^3_{o\gamma\gamma}.[[IncreasingF(F,\lhd^1,\lhd^2) \wedge DecreasingF(G,\lhd^2,\lhd^3)] \Rightarrow DecreasingF(F \circ G, \lhd^1, \lhd^3)]$
752+4	$\forall X_{o\alpha}, Y_{o\alpha}, F_{\beta\alpha}.[ImageF(F, X \cup Y) = ImageF(F,X) \cup ImageF(F,Y)]$
753+4	$\forall X_{o\alpha}, Y_{o\alpha}, F_{\beta\alpha}.[ImageF(F, X \cap Y) \subseteq ImageF(F,X) \cap ImageF(F,Y)]$
764+4	$\forall F_{\beta\alpha}.[InverseImageF(F,\emptyset) = \emptyset]$
770+4	$\forall R_{o\beta\alpha}, Q_{o\beta\alpha}.[[EquivRel(R) \wedge EquivRel(Q)] \Rightarrow [EquivClasses(R) = EquivClasses(Q) \vee Disjoint(EquivClasses(R), EquivClasses(Q))]]$

clause normalisation rules to reach some proper initial clauses. In our concrete case, this normalisation process leads to the following unit clause consisting of a (syntactically not solvable) unification constraint (here $B_{o\alpha}, C_{o\alpha}, D_{o\alpha}$ are Skolem constants and Bx is obtained from expansion of $x \in B$):

$$[(\lambda x_\alpha.Bx \vee (Cx \wedge Dx)) =^? (\lambda x_\alpha.(Bx \vee Cx) \wedge (Bx \vee Dx))]$$

Note that negated primitive equations are generally automatically converted by Leo into unification constraints. This is why $[(\lambda x_\alpha.Bx \vee (Cx \wedge Dx)) =^?$

Table 2. Defined concepts occurring in problems from Table 1

Defined Notions in Theory Typed Set		
$\_ \in \_$	$:=$	$\lambda x_\alpha, A_{o\alpha}.[Ax]$
$\emptyset$	$:=$	$[\lambda x_\alpha.\bot]$
$\_ \subseteq \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\forall x_\alpha.x \in A \Rightarrow x \in B]$
$\_ \cup \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_\alpha.x \in A \vee x \in B]$
$\_ \cap \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_\alpha.x \in A \wedge x \in B]$
$\overline{\_}$	$:=$	$\lambda A_{o\alpha}.[\lambda x_\alpha.x \notin A]$
$\_ \backslash \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_\alpha.x \in A \wedge x \notin B]$
$ExclUnion(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[(A \backslash B) \cup (B \backslash A)]$
$Disjoint(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[A \cap B = \emptyset]$
$Meets(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\exists x_\alpha.x \in A \wedge x \in B]$
$Misses(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\neg\exists x_\alpha.x \in A \wedge x \in B]$
Defined Notions in Theory Relation		
$UnOrderedPair(\_,\_)$	$:=$	$\lambda x_\alpha, y_\alpha.[\lambda u_\alpha.u = x \vee u = y]$
$Singleton(\_)$	$:=$	$\lambda x_\alpha.[\lambda u_\alpha.u = x]$
$Pair(\_,\_)$	$:=$	$\lambda x_\alpha, y_\beta.[\lambda u_\alpha, v_\beta.u = x \wedge v = y]$
$\_ \times \_$	$:=$	$\lambda A_{o\alpha}, B_{o\beta}.[\lambda u_\alpha, v_\beta.u \in A \wedge v \in B]$
$RDom(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\lambda x_\alpha.\exists y_\beta.Rxy]$
$RCodom(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\lambda y_\beta.\exists x_\alpha.Rxy]$
$Subrel(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, Q_{o\beta\alpha}.[\forall x_\alpha.\forall y_\alpha.Rxy \Rightarrow Qxy]$
$Id(\_)$	$:=$	$\lambda A_{o\alpha}.[\lambda x_\alpha, y_\alpha.x \in A \wedge x = y]$
$Field(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[RDom(B) \cup RCodom(R)]$
$IsRelOn(\_,\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, A_{o\alpha}.\lambda B_{o\beta}.[\forall x_\alpha, y_\beta.Rxy \Rightarrow (x \in A \wedge x \in B)]$
$RestrictRCodom(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, A_{o\alpha}.[\lambda x_\alpha, y_\beta.x \in A \wedge Rxy]$
$RelComp(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, Q_{o\gamma\beta}.[\lambda x_\alpha, z_\gamma.\exists y_\beta.Rxy \wedge Ryz]$
$InverseImageR(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, B_{o\beta}.[\lambda x_\alpha.\exists y_\beta.y \in B \wedge Rxy]$
$Reflexive(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\forall x_\alpha.Rxx]$
$Symmetric(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\forall x_\alpha.\forall y_\alpha.Rxy \Rightarrow Ryx]$
$Transitive(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\forall x_\alpha.\forall y_\alpha.\forall z_\alpha.Rxy \wedge Ryz \Rightarrow Rxz]$
$EquivRel(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[Reflexive(R) \wedge Symmetric(R) \wedge Transitive(R)]$
$EquivClasses(\_)$	$:=$	$\lambda R_{o\alpha\alpha}.[\lambda A_{o\alpha}.\exists u_\alpha.u \in A \wedge \forall v_\alpha.v \in A \Leftrightarrow Ruv]$
Defined Notions in Theory Function		
$Inj(\_)$	$:=$	$\lambda F_{\beta\alpha}.[\forall x_\alpha, y_\beta.F(x) = F(y) \Rightarrow x = y]$
$Surj(\_)$	$:=$	$\lambda F_{\beta\alpha}.[\forall y_\beta.\exists x_\alpha.y = F(x)]$
$Bij(\_)$	$:=$	$\lambda F_{\beta\alpha}.Surj(F) \wedge Inj(F)$
$ImageF(\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, A_{o\alpha}.[\lambda y_\beta.\exists x_\alpha.x \in A \wedge y = F(x)]$
$InverseImageF(\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, B_{o\beta}.[\lambda x_\alpha.\exists y_\beta.y \in B \wedge y = F(x)]$
$\_ \circ \_$	$:=$	$\lambda F_{\beta\alpha}, G_{\gamma\beta}.[\lambda x_\alpha.G(F(x))]$
$IncreasingF(\_,\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, \lhd^1_{o\alpha\alpha}, \lhd^2_{o\beta\beta}.[\forall x_\alpha, y_\alpha.x \lhd^1 y \Rightarrow F(x) \lhd^2 F(y)]$
$DecreasingF(\_,\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, \lhd^1_{o\alpha\alpha}, \lhd^2_{o\beta\beta}.[\forall x_\alpha, y_\alpha.x \lhd^1 y \Rightarrow F(y) \lhd^2 F(x)]$

$(\lambda x_\alpha.(Bx \vee Cx) \wedge (Bx \vee Dx))]$ is generated, and not $[(\lambda x_\alpha.Bx \vee (Cx \wedge Dx)) = (\lambda x_\alpha.(Bx \vee Cx) \wedge (Bx \vee Dx))]^F$. 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 this unification constraint by the negative literal (where x is a Skolem constant):

$$[(Bx \vee (Cx \wedge Dx)) \Leftrightarrow ((Bx \vee Cx) \wedge (Bx \vee Dx))]^F$$

This unit clause is again not normal; normalisation, factorisation and subsumption yield the following set of clauses:

$$[Bx]^F \qquad [Bx]^T \vee [Cx]^T \qquad [Bx]^T \vee [Dx]^T \qquad [Cx]^F \vee [Dx]^F$$

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.

Let us consider now this same example SET171+3 in its first-order formulation from the TPTP (see Table 3). We can observe that the assumptions provide

Table 3. TPTP problem SET171+3 — distributivity of $\cup$ over $\cap$

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

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. [20], 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 [14] (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.

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[4] 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

[4] 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.

embedded equations), and are therefore suitable for treatment by a first-order ATP or even a propositional logic decision procedure.

SET624+3 Sometimes, extensionality treatment is not required and the originally higher-order problem is immediately reduced to only FO-like clauses. For example, after expanding the definitions, problem SET624+3 yields the following clause (where $B_{o\alpha}, C_{o\alpha}, D_{o\alpha}$ are again Skolem constants):

$$[(\exists x_\alpha . (Bx \wedge (Cx \vee Dx)) \Leftrightarrow ((\exists x_\alpha . Bx \wedge Cx) \vee (\exists x_\alpha . Bx \wedge Dx))]^F$$

Normalisation results in 26 FO-like clauses, which present a hard problem for Leo: it needs approx. 35 seconds (see Sec. 4) to find a refutation, whereas first-order ATPs only need a fraction of a second.

SET646+3 Sometimes, problems are immediately refuted after the initial clause normalisation. For example, after definition expansion in problem SET646+3 we get the following clause (where $B_{o\alpha}, C_{o\alpha}, x_\alpha$ are again Skolem constants):

$$[Ax \Rightarrow (\forall y_\beta . By \Rightarrow (\forall u_\alpha . \forall v_\beta . (u = x \wedge v = y) \Rightarrow ((\neg\bot) \wedge (\neg\bot))))]^F$$

Normalisation in Leo immediately generates a basic refutation (i.e., a clause $[\bot]^T \vee [\bot]^T$) without even starting proof search.

SET611+3 The examples discussed so far all essentially apply extensionality treatment and normalisation to the input problem in order to immediately generate a set of inconsistent FO-like clauses. Problem SET611+3 is more complicated as it requires several reasoning steps in Leo before the initially consistent set of available FO-like clauses grows into an inconsistent one. After definition expansion, Leo is first given the input clause:

$$[\forall A_{o\alpha}, B_{o\alpha} . (\lambda x_\alpha . (Ax \wedge Bx)) = (\lambda x_\alpha . \bot)) \Leftrightarrow (\lambda x_\alpha . (Ax \wedge \neg Bx)) = (\lambda x_\alpha . Ax)]^F$$

which it normalises into:

$$[(\lambda x_\alpha . (Ax \wedge Bx)) =^? (\lambda x_\alpha . \bot)] \vee [(\lambda x_\alpha . (Ax \wedge \neg Bx)) =^? (\lambda x_\alpha . Ax)] \quad (9)$$

$$[(\lambda x_\alpha . (Ax \wedge Bx)) = (\lambda x_\alpha . \bot)]^T \vee [(\lambda x_\alpha . (Ax \wedge \neg Bx)) = (\lambda x_\alpha . Ax)]^T \quad (10)$$

As mentioned before, the unification constraint (9) corresponds to:

$$[(\lambda x_\alpha . (Ax \wedge Bx)) = (\lambda x_\alpha . \bot)]^F \vee [(\lambda x_\alpha . (Ax \wedge \neg Bx)) = (\lambda x_\alpha . Ax)]^F \quad (11)$$

Leo has to apply to each of these clauses and to each of their literals appropriate extensionality rules. Thus, several rounds of Leo's set-of-support-based reasoning procedure are required, so that all necessary extensionality reasoning steps are performed, and sufficiently many FO-like clauses are generated which can be refuted by Bliksem.

In summary, each of the examples discussed in this section exposes a motivation for our higher-order/first-order cooperative approach to theorem proving. In particular, they show that:

- Higher-order formulations allow for a concise problem representation which often allows easier and faster proof search than first-order formulations.
- Higher-order problems can often be reduced to a set of first-order clauses that can be more efficiently handled by a first-order ATP.
- Some problems are trivially refutable after clause normalisation.
- Some problems require in-depth higher-order reasoning before a refutable first-order clause set can be extracted.

3 Higher-Order/First-Order Cooperation via OANTS

The cooperation between higher-oder and first-order reasoners, which we investigate in this paper, is realised in the concurrent hierarchical blackboard architecture OANTS [7]. We first describe in Sec. 3.1 the existing OANTS architecture. In order to overcome some of its problems, in particular efficiency problems, we devised within OANTS a new and improved cooperation method for the higher-order ATP LEO and first-order provers (in particular, BLIKSEM) – we describe this in Sec. 3.2. We address the question of how to generate the necessary clauses in Sec. 3.3, and discuss soundness and completeness of our implementation of the higher-order/first-order cooperation in Sec. 3.4.

3.1 OANTS

OANTS was originally conceived to support interactive theorem proving but was later extended to a fully automated proving system [22,8]. 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 OANTS 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.

OANTS employs resource reasoning to guide search.[5] This enables the controlled integration (e.g., by specifying time-outs) of full-fledged external reasoning systems such as automated theorem provers, computer algebra systems, or model generators into the architecture. 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 application of an ATP has its conclusion argument set to be an open goal. A process can then place an open goal on the blackboard, where it is picked up by a process that applies the prover to it. Any computed proof or partial-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. Since there are many types of integrated systems, the language of the proof object — 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 (large clause sets need to be transformed into large single formulae to represent them in the proof object; the support for this in OANTS to date is inefficient), even if the reasoning of all systems involved is clause-based. 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 OANTS. 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 OANTS as a single inference rule. The cooperation between these two theorem provers is carried out directly and not via the central proof object. This avoids translating clause sets into single formulae and back. While in our previous approach the cooperation between LEO and an FO-ATP was modelled at the upper layer of the OANTS architecture, our new approach presented in this paper models their cooperation by exploiting the lower layer of the OANTS blackboard architecture. This is not an ad hoc solution,

[5] OANTS 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.

but rather, it demonstrates OANTS's flexibility in modelling the integration of cooperative reasoning systems.

Concretely, the single inference rule modelling the cooperation between LEO and a first-order theorem prover needs four arguments to be applicable: (1) an open 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. 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 can be applied and the open proof goal can be closed in the central proof object. That is, the open goal can be 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.

The cooperation can be carried out between any first-order theorem prover and LEO instantiated with any strategy, thus resulting in different instantiations of the inference rule discussed above. While several first-order provers are integrated in OANTS and could be used, BLIKSEM was sufficient for the case study reported in this paper (see Sec. 4). In most cases, more than one BLIKSEM process was necessary. But as the problems were always concerned with only one subgoal, only one LEO process had to be started.

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 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. A disadvantage of this 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 OANTS, as is possible when applying only one prover per open subgoal. Providing such translation remains future work. The repercussions will be discussed in more detail in Sec. 3.4.

3.3 Extracting FO-Like Clauses from LEO

Crucial to a successful cooperation between LEO and a first-order ATP is obviously the generation of FO-like clauses. LEO always maintains a heap of FO-like clauses. In the current LEO system this heap remains rather small since LEO's standard calculus intrinsically avoids primitive equality and instead provides a rule that replaces occurrences of primitive equality with their corresponding Leibniz definitions which are higher-order. The Leibniz principle defines equality as follows $=_{o\alpha\alpha} := \lambda x_\alpha . \lambda y_\alpha . [\forall P_{o\alpha} . Px \Rightarrow Py]$. LEO also provides a rule which replaces syntactically non-unifiable unification constraints between terms of non-Boolean base type by their respective representations that use Leibniz equality. While the clauses resulting from these rules are still refutable in LEO, they are not refutable by BLIKSEM without adding set theory axioms. We illustrate the effect by the following simple example, where a_ι, b_ι, and $f_{\iota\iota}$ are constants:

$$a = b \Rightarrow f(a) = f(b)$$

Depending on whether we work with primitive equality or Leibniz equality this problem is reduced to the clause sets in either (12) or (13) respectively (in the latter $\mathbf{P}_{o\iota}$ is a new free variable, and $Q_{o\iota}$ is a new Skolem constant):

$$[a = b]^T \qquad [f(a) =^? f(b)] \tag{12}$$

$$[\mathbf{P}a]^F \vee [\mathbf{P}b]^T \qquad [Q(f(a))]^T \qquad [Q(f(b))]^F \tag{13}$$

While the former is obviously refutable in BLIKSEM, the latter is not. LEO, however, still finds a refutation for the latter and generates the crucial substitution $\mathbf{P} \leftarrow \lambda x_\alpha . Q(f(x))$ by higher-order pre-unification.

To circumvent this problem, we adapted the relevant rules in LEO. Instead of immediately constructing Leibniz representation of clauses, an intermediate representation containing primitive equality is generated and dumped on the heap of FO-like clauses. As a consequence, additional useful FO-like clauses are accumulated and the heap can become quite large, in particular, since we do not apply any subsumption to the set of FO-like clauses (this is generally done more efficiently by a first-order ATP anyway). Recent research has shown that Leibniz equality is generally very bad for automating higher-order proof search. Thus, future work in LEO includes providing support for full primitive equality and avoiding Leibniz equations.

3.4 Soundness and Completeness of the Cooperation

Clearly, soundness and completeness properties depend on the corresponding properties of the systems involved, in our case, of LEO and BLIKSEM.

Soundness: The general philosophy of OANTS is to ensure the correctness of proofs by the generation of explicit proof objects, which can be checked independently from the proof generation. In particular, reasoning steps of ATPs have to be translated into OANTS's natural deduction calculus via the TRAMP proof

transformation system [17] to be machine-checkable. Since the cooperative proof result of Leo-Bliksem cannot yet be directly inserted into the centralised proof object, the generation of a machine-checkable proof object is not yet supported. One possible solution is to insert Bliksem proofs into Leo proofs at the right places. Then, the modified Leo proofs can be inserted into the centralised proof object, and hence, explicit proof objects can be generated by Oants. In principle, there is no problem with this, however, it is not yet implemented.

While there are many advantages in guaranteeing correctness of proofs by checking them, it is worth noting that the combination of Leo and Bliksem is sound under the assumption that the two systems are sound. Namely, to prove a theorem it is sufficient to show that a subset of clauses generated in the proof is inconsistent. If Leo generates an inconsistent set of clauses, then it does so correctly by assumption, be it a FO-like set or not. Assuming that the translation from FO-like clauses to truly first-order clauses preserves consistency/inconsistency, then a set of clauses that is given to Bliksem is inconsistent only if Leo generated an inconsistent set of clauses in the first place. By the assumption that Bliksem is sound follows that Bliksem will only generate the empty clause when the original clause set was inconsistent.

Thus, soundness of our cooperative approach critically relies only on the soundness of the selected transformational mapping from FO-like clauses to proper first-order clauses. We use the mapping from Tramp, which has been previously shown to be sound and is based on [16]. Essentially, it injectively maps expressions such as $P(f(a))$ to expressions such as $@^1_{\text{pred}}(P, @^1_{\text{fun}}(f, a))$, where the @ are new first-order operators describing function and predicate application for particular types and arities. The injectivity of the mapping guarantees soundness, since it allows each proof step to be mapped back from first-order to higher-order. Hence, our higher-order/first-order cooperative approach between Leo and Bliksem is sound.

Completeness: Completeness (in the sense of Henkin completeness) can in principle be achieved in higher-order systems, but practically, the strategies used are typically not complete for efficiency reasons. Let us assume that we use a complete strategy in Leo. All that our procedure does is pass FO-like clauses to Bliksem. Hence, no proofs can be lost in this process. That is, completeness follows trivially from the completeness of Leo.

The more interesting question is whether particular cooperation strategies will be complete as well. For instance, in Leo we may want to give higher preference to real higher-order steps which guarantee the generation of first-order clauses.

4 Experiments and Results

We conducted several experiments to evaluate our hybrid reasoning approach. In particular, we concentrated on problems given in Table 1. We investigated several Leo strategies in order to compare Leo's individual performance with the performance of the Leo-Bliksem cooperation. Our example set differs from

the one in [14] in that it contains some additional problems, and it also omits an entry for problem SET108+1. This problem addresses the universal class and can therefore not be formalised in type theory in the same concise way as the other examples, but only in a way very similar to the one given in TPTP.

Table 4 presents the results of our experiments. All timings given in the table are in seconds. The first column contains the TPTP identifier of the problem. The second column relates some of the problems to their counterparts in the Journal of Formalized Mathematics (JFM; see `mizar.org/JFM`) where they originally stem from. This eases the comparison with the results in [6,2], where the problems from the JFM article *Boolean Properties of Sets* were already solved: the problems are named with prefix 'B:'. Prefix 'RS1:' stands for the JFM article *Relations Defined on Sets*. The third column lists the TPTP (v3.0.1 as of 20 January 2005, see `http://www.tptp.org`) difficulty rating of the problem, which indicates how hard the problem is for first-order ATPs (difficulty rating 1.00 indicates that no TPTP prover can solve the problem).

The fourth, fifth and sixth columns list whether SATURATE, MUSCADET (v2.4) and E-SETHEO (csp04), respectively, can (+) or cannot (–) solve a problem. The seventh column lists the timing results for VAMPIRE (v7). The results for SATURATE are taken from [14] (a '?' in Table 4 indicates that the result was not listed in [14] and is thus unavailable). The results for MUSCADET and E-SETHEO are taken from the on-line version of the solutions provided with the TPTP. Since the listed results were obtained from different experiments on different platforms, their run-time comparison would be unfair, and was thus not carried out. The timings for VAMPIRE, on the other hand, are based on private communication with A. Voronkov and they were obtained on a computer with a very similar specification as we used for the LEO-BLIKSEM timings. Note, that the results for VAMPIRE and E-SETHEO reported in [14] differ for some of the problems to the ones in TPTP. This is probably due to different versions of the systems tested, for instance, the TPTP uses VAMPIRE version 7, while the results reported in [14] are based on version 5. The results in columns four through to seven show that some problems are still very hard for first-order ATPs, as well as for the special purpose theorem prover MUSCADET. Column eight and nine in Table 4 list the results for LEO alone and LEO-BLIKSEM, respectively. Each of these two columns is further divided into sub-columns to allow for a detailed comparison.

All our experiments (for the values of LEO and LEO-BLIKSEM) were conducted on a 2.4 GHz Xenon machine with 1GB of memory and an overall time limit of 100 seconds. For our experiments with LEO alone in column eight in Table 4 we tested four different strategies. Mainly, they differ in their treatment of equality and extensionality. This ranges from immediate expansion of primitive equality with Leibniz equality and limited extensionality reasoning, STANDARD (ST), to immediate expansion of primitive equality and moderate extensionality reasoning, EXT, to delayed expansion of primitive equality and moderate extensionality reasoning, EXT-INPUT (EI), and finally to delayed expansion of primitive equality and advanced recursive extensionality reasoning,

Table 4. Experimental data for the benchmark problems given in Table 1

TPTP-Problem	Mizar Problem	Difficulty	Saturate	Mus cadet	E-Setheo	Vampire 7	LEO Strat.	LEO Cl.	LEO Time	LEO-Bliksem Cl.	LEO-Bliksem Time	LEO-Bliksem FOcl	LEO-Bliksem FOtm	LEO-Bliksem GnCl
SET014+4		.67	+	+	+	.01	ST	41	.16	34	6.76	19	.01	7
SET017+1		.56	–	–	+	.03	EXT	3906	57.52	25	8.54	16	.01	74
SET066+1		1.00	?	–	–	–	–	–	–	26	6.80	20	10	56
SET067+1		.56	+	+	+	.04	ST	6	.02	13	.32	16	.01	12
SET076+1		.67	+	–	+	.00	–	–	–	10	.47	18	.01	35
SET086+1		.22	+	–	+	.04	ST	4	.01	4	.01	N/A	N/A	N/A
SET096+1		.56	+	–	+	.03	–	–	–	27	7.99	14	.01	25
SET143+3	B:67	.67	+	+	+	68.71	EIR	37	.38	33	7.93	18	.01	19
SET171+3	B:71	.67	+	+	–	108.31	EIR	36	.56	25	4.75	19	.01	20
SET580+3	B:23	.44	+	+	+	14.71	EIR	25	.19	6	2.73	8	.01	13
SET601+3	B:72	.22	+	+	+	168.40	EIR	145	2.20	55	4.96	8	.01	13
SET606+3	B:77	.78	+	–	+	62.02	EIR	21	.33	17	10.8	15	.01	5
SET607+3	B:79	.67	+	+	+	65.57	EIR	22	.31	17	7.79	15	.01	6
SET609+3	B:81	.89	+	+	–	161.78	EIR	37	.60	26	6.50	19	10	17
SET611+3	B:84	.44	+	–	+	60.20	EIR	996	12.69	72	32.14	38	.01	101
SET612+3	B:85	.89	+	–	–	113.33	EIR	41	.54	18	3.95	6	.01	7
SET614+3	B:88	.67	+	+	–	157.88	EIR	38	.46	19	4.34	16	.01	17
SET615+3	B:89	.67	+	+	–	109.01	EIR	38	.57	17	3.59	6	.01	9
SET623+3	B:99	1.00	?	–	–	–	EXT	43	8.84	23	9.54	10	.01	14
SET624+3	B:100	.67	+	–	+	.04	ST	4942	34.71	54	9.61	46	.01	212
SET630+3	B:112	.44	+	–	+	60.39	EIR	11	.07	6	.08	8	10	4
SET640+3	RS1:2	.22	+	–	+	70.41	EIR	2	.01	2	.01	N/A	N/A	N/A
SET646+3	RS1:8	.56	+	–	+	59.63	EIR	2	.01	2	.01	N/A	N/A	N/A
SET647+3	RS1:9	.56	+	–	+	64.21	EIR	26	.15	13	.30	13	.01	15
SET648+3	RS1:10	.56	+	–	+	64.22	EIR	26	.15	14	.30	13	.01	16
SET649+3	RS1:11	.33	–	–	+	63.77	EIR	45	.30	29	5.49	12	.01	16
SET651+3	RS1:13	.44	–	–	+	63.88	EIR	20	.10	11	.16	10	10	11
SET657+3	RS1:19	.67	+	–	+	1.44	EIR	2	.01	2	.01	N/A	N/A	N/A
SET669+3	RS1:19	.22	–	–	+	.34	EI	35	.22	35	.23	N/A	N/A	N/A
SET670+3	RS1:33	1.00	?	–	–	–	EXT	15	.17	17	.36	16	.01	6
SET671+3	RS1:34	.78	–	–	+	218.02	EIR	78	.64	7	2.71	10	.01	14
SET672+3	RS1:35	1.00	?	–	–	–	EXT	27	.4	30	.70	21	.01	11
SET673+3	RS1:36	.78	–	–	+	47.86	EIR	78	.65	14	5.66	14	.01	16
SET680+3	RS1:47	.33	+	–	+	.07	ST	185	.88	29	4.61	18	.01	24
SET683+3	RS1:50	.22	+	–	+	.06	ST	46	.20	35	8.90	18	10	24
SET684+3	RS1:51	.78	–	–	+	.33	ST	275	2.45	46	5.95	26	.01	47
SET686+3	RS1:53	.56	–	–	+	.11	ST	274	2.36	46	5.37	26	.01	46
SET716+4		.89	+	+	–	–	ST	39	.45	18	3.81	18	.01	118
SET724+4		.89	+	+	–	–	EXT	154	2.75	18	7.21	15	10	23
SET741+4		1.00	?	–	–	–	–	–	–	–	–	–	–	–
SET747+4		.89	–	+	–	–	ST	34	.46	25	1.11	18	10	10
SET752+4		.89	?	+	–	–	–	–	–	50	6.60	48	.01	4363
SET753+4		.89	?	+	–	–	–	–	–	15	3.07	12	10	19
SET764+4		.56	+	+	+	.02	EI	9	.05	8	.04	N/A	N/A	N/A
SET770+4		.89	+	+	–	–	–	–	–	–	–	–	–	–

EXT-INPUT-RECURSIVE (EIR). Column eight in Table 4 presents the fastest strategy for a respective problem (Strat.), the number of clauses generated by Leo (Cl.), and the total runtime (Time). While occasionally there were more than one Leo strategy that could solve a problem, it should be noted that none of the strategies was successful for all the problems solved by Leo.

In contrast to the experiments with Leo alone, we used only the EXT-INPUT strategy for our experiments with the Leo-Bliksem cooperation. Column nine in Table 4 presents the number of clauses generated by Leo (Cl.) together with the time (Time), and in addition, the number of first-order clauses sent to Bliksem (FOcl), the time used by Bliksem (FOtm), and the number of clauses generated

by Bliksem (GnCl). Note, that we give the data only for the first instance that Bliksem actually succeeded in solving the problem. This time also includes the time needed to write and process input and output files over the network. While Leo and instances of Bliksem were running in separate threads (each run of Bliksem was given a 50 second time limit), the figures given in the 'Time' column reflect the overall time needed for a successful proof. That is, it contains the time needed by all concurrent processes: Leo's own process as well as those processes administering the various instances of Bliksem. Since all these processes ran on a single processor, there is potential to ameliorate the overall runtimes by using real multiprocessing.

Note also, that the number of clauses in Leo's search space is typically low since subsumption is enabled. Subsumption, however, was not enabled for the accumulation of FO-like clauses in Leo's bag of FO-like clauses. This is why there are usually more clauses in this bag (which is sent to Bliksem) than there are available in Leo's search space. Finally, observe that for some problems a refutation was found after Leo's clausal normalisation, and therefore Bliksem was not applicable (N/A).

While Leo itself can solve a majority of the considered problems with some strategy, the Leo-Bliksem cooperation can solve more problems and, moreover, needs only a single Leo strategy. We can also observe that for many problems that appear to be relatively hard for Leo alone (e.g., SET017+1, SET611+3, SET624+3), the Leo-Bliksem cooperation solves them not only more quickly, but also it sometimes reduces the problems to relatively small higher-order preprocessing steps with subsequent easy first-order proofs, as for instance, in the case of SET017+1.

From a mathematical viewpoint the investigated problems are trivial and, hence, they should ideally be reliably and very efficiently solvable within a proof assistant. This has been achieved for the examples in Table 4 (except for SET741+4 and SET770+4) by our hybrid approach. While some of the proof attempts now require slightly more time than when using Leo alone with a specialised strategy, they are, in most cases, still faster than when proving with a first-order system.

5 Related Work and Conclusion

Related to our approach is the Techs system [12], 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 [13]) 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 [15] which realises a generic interface between HOL and first-order theorem provers. It is similar to the solution

previously achieved by TRAMP [17] in OMEGA, which serves as a basis for the sound integration of ATPs into OANTS. 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 [18]. 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 [15,17,18].

One of the motivations for our work is to show that the cooperation of higher-order and first-order automated theorem provers can be very successful and effective. The results of our case study provide evidence for this: our non-optimised system outperforms related work on state-of-the-art first-order theorem provers and their ad hoc extensions such as SATURATE [14] on 45 mathematical problems chosen from the TPTP SET category. Among them are four problems which cannot be solved by any TPTP system to date. In contrast to the first-order situation, these problems can in fact be proved in our approach reliably from first principles, that is, without avoiding relevant base axioms of the underlying set theory, and moreover, without the need to provide *relevant* lemmata and definitions by hand.

The results of our case study motivate further research in the automation of higher-order theorem proving and the experimentation with different higher-order to first-order transformation mappings (such as the ones used by Hurd) that support our hybrid reasoning approach. They also provide further evidence for the usefulness of the OANTS approach as described in [8,5] for flexibly modelling the cooperation of reasoning systems.

Our results also motivate the need for a higher-order extension of the TPTP library in which alternative higher-order problem formalisations are linked with their first-order counterparts so that first-order theorem provers could also be evaluated against higher-order systems (and vice versa).

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 [9] and Brown's PhD thesis on *set comprehension in Church's type theory* [10].

Acknowledgements For advice and help we thank Chad Brown, Andreas Meier, Andrei Voronkov, and Claus-Peter Wirth.

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