Available online at www.sciencedirect.com
ScienceDirect
Journal of Applied Logic 6 (2008) 318–342
JOURNAL OF APPLIED LOGIC
www.elsevier.com/locate/jal

Combined reasoning by automated cooperation ☆

Christoph Benzmüller [a,b], Volker Sorge [c], Mateja Jamnik [a], Manfred Kerber [c,*]

[a] *University of Cambridge, Computer Laboratory, Cambridge, England, UK*
[b] *Informatik, Universität des Saarlandes, Saarbrücken, Germany*
[c] *Computer Science, University of Birmingham, Birmingham, England, UK*

Received 8 May 2007; received in revised form 12 June 2007; accepted 19 June 2007

Available online 27 June 2007

Abstract

Different reasoning systems have different strengths and weaknesses, and often it is useful to combine these systems to gain as much as possible from their strengths and retain as little as possible from their weaknesses. Of particular interest is the integration of first-order and higher-order techniques. First-order reasoning systems, on the one hand, have reached considerable strength in some niches, but in many areas of mathematics they still cannot reliably solve relatively simple problems, for example, when reasoning about sets, relations, or functions. Higher-order reasoning systems, on the other hand, can solve problems of this kind automatically. But the complexity inherent in their calculi prevents them from solving a whole range of problems. However, while many problems cannot be solved by any one system alone, they can be solved by a combination of these systems.

We present a general agent-based methodology for integrating different reasoning systems. It provides a generic integration framework which facilitates the cooperation between diverse reasoners, but can also be refined to enable more efficient, specialist integrations. We empirically evaluate its usefulness, effectiveness and efficiency by case studies involving the integration of first-order and higher-order automated theorem provers, computer algebra systems, and model generators.

Keywords: Automated reasoning; System integration; Agent architecture; Higher-order and first-order proving; Computer algebra; Model generators

1. Introduction

The last decade has seen the development of various reasoning systems which are specialised in specific problem domains. Theorem proving contests, such as the annual CASC[1] competition, have shown that these systems typically perform well in particular niches but often do poorly in others. First-order provers, for instance, cannot be used to prove higher-order problem formulations. Deduction systems in general are very weak in carrying out computations,

☆ This work was supported by EPSRC grant GR/M22031 and DFG-SFB 378 (C. Benzmüller), EU Marie-Curie-Fellowship HPMF-CT-2002-01701 (V. Sorge), and EPSRC Advanced Research Fellowship GR/R76783 (M. Jamnik).
* Corresponding author.
E-mail address: m.kerber@cs.bham.ac.uk (M. Kerber).
URLs: http://www.ags.uni-sb.de/~chris (C. Benzmüller), http://www.cs.bham.ac.uk/~vxs (V. Sorge), http://www.cl.cam.ac.uk/users/mj201 (M. Jamnik), http://www.cs.bham.ac.uk/~mmk (M. Kerber).
[1] **C**ADE **A**TP **S**ystem **C**ompetitions, see also http://www.tptp.org.

1570-8683/$ – see front matter
doi:10.1016/j.jal.2007.06.003

whereas computer algebra systems are strong in algebraic computations (e.g., derivatives of functions) but weak at finding logical arguments. Previous work, such as [24], tackled this problem by integrating specific reasoning systems, but typically this integration has been hard-wired and also special purpose. Only rather few architectures have been discussed so far that try to extend the application range and hence the generality of reasoning systems by a flexible integration of different specialist systems. Some such related systems are further discussed in Section 5.

The aim of our work is to broaden the range of mechanisable mathematics by allowing a flexible cooperation between specialist systems. Thus, we developed a framework for such integration of a variety of reasoning systems by using an agent-oriented approach, and we present it in this paper. Some of our ideas and first implementations of this framework were reported previously in [6,7,11]. An agent-architecture offers several benefits in developing such a framework. From a software engineering point of view it provides a simple and flexible methodology for integrating systems. Furthermore, it enables a flexible proof search where each single system—in form of a proactive software agent—can focus on parts of the problem it is good at, without the need for specifying *a priori* a hierarchy of their application.

In our framework we employ a centralised approach and focus on the construction of a single proof object. This means that all agents pick up and investigate the central proof object that is given in higher-order natural deduction style with additional facilities to abstract from the pure calculus layer [16]. If an agent determines that it is applicable in the current proof context, then it carries out its task, for instance, by invoking a tactic or by calling the external system it encapsulates. The agent is given a fixed amount of resources for this, and when these are consumed it comes back and makes bids in terms of a modified proof object. Based on heuristic criteria, one bid is accepted and executed by the central system while the remaining ones are stored for backtracking purposes. In this sense, we use our central proof object for establishing global cooperation and communication. The benefit is that only translations into a single proof representation language are required, which reduces the proof theoretical and logical issues to be addressed. Furthermore, our central proof object uses a human-oriented natural deduction format which eases user interaction. We discuss the agent-framework for generic integration of reasoning systems in more detail in Section 2.

Our current system, which implements the agent-framework just outlined, combines different reasoning components such as specialised state-of-the-art higher-order and first-order theorem provers, model generators, and computer algebra systems. It employs a classical natural deduction calculus and tactical theorem proving in the background to bridge the gaps between different subproofs of the individual components, as well as to guarantee correctness of constructed proofs. In Section 3, we present some case studies by running examples of how theorems from a variety of domains are tackled in our system.

While our generic framework for integrating a variety of systems is indeed very flexible and generic, a centralised approach can lead to excessive communication between the agents. This is a weakness of our generic approach. Hence, we studied in detail how some fine-tuning, which decentralises communication, can improve efficiency of cooperation between the systems. In particular, in Section 4 we present this specialist integration for first-order and higher-order theorem provers. The experiments that we carried out with this implementation show that decentralising communication and using specialist integration for particular systems leads to a much higher overall efficiency and power of the central system and enables it to solve problems that are still very hard for first-order automated theorem provers.

2. Agent architecture for integrating reasoning systems

Here we describe our multi-agent framework for flexibly integrating any number of diverse reasoning systems. We start in Section 2.1 with the generic architecture, which provides a simple mechanism for incorporating new reasoning systems. The generality feature of the architecture is in some cases traded-off for less than optimal performance, and we therefore present in Section 2.2 a refinement of the generic approach in order to optimise its performance.

The cooperation between the different systems of the framework is realised in the concurrent hierarchical blackboard architecture OANTS [9], in which different reasoning systems can pick up problems (or subproblems) from a blackboard and contribute to the overall solution by solving problems or generating subproblems. OANTS was originally conceived to support interactive theorem proving but was later extended to a fully automated proving system [10,36]. Its basic idea is to compose a *central proof object* by generating, in each proof situation, a ranked list of bids 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. These applicability checks are performed by separate processes, i.e. software agents which compute and report bids.

2.1. A two-layered architecture

The architecture consists of two layers. On the lower layer, bids of possible instantiations of the arguments of individual inference rules are computed. In particular, each inference rule is associated with its own blackboard and several concurrent processes, at least one process for each argument of the inference rule. The role of every process is to compute bids of possible instantiations for its designated argument of the inference rule, and to record these bids on the blackboard for this rule. The computations are carried out with respect to the given proof context and by exploiting bids already present on the rules' blackboard, that is, argument instantiations computed by other processes working for the same rule. On the upper layer, the bids from the lower layer that are applicable in the current proof state are accumulated and heuristically ranked by another process. For instance, bids with closed (sub)goals are preferred over partial results, and big steps in the search space are preferred over calculus level steps. The most promising bid on the upper layer 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-bounded reasoning to guide search.[2] 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. The semantics of the rules connecting to external reasoners is currently hand-coded. Further work includes to investigate whether an ontology as suggested in [38] could be fruitfully employed.

The advantage of this setup is that it enables proof construction by a collaborative effort of diverse reasoning systems. Moreover, the architecture provides a simple and general mechanism for integrating new reasoners in the system. The integration is independent of all the other systems already integrated. Adding a new reasoning system to the architecture requires only transforming the new system into an agent by wrapping a shell around it. This shell can communicate with the blackboard by reading and writing subproblems to and from the blackboard, as well as writing proofs back to the blackboard in a standardised format.

The disadvantage of such a generic architecture that employs centralised communication is that the cooperation can be achieved only 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 very rich higher-order language, 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 (e.g., 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 [7]: the larger part of the proof effort may in some cases be spent on communication rather than on proof search.

In summary, while central communication eases cooperation between any type of reasoner, certain systems could communicate far more efficiently with each other using dedicated formalisms. To exploit this fact—and thus overcome the communication bottleneck—we devised a new method for the cooperation between two integrated systems via a single inference rule, which we describe in detail next.

[2] OANTS also provides facilities to define and modify the processes at run-time, but we do not use these advanced features in the case studies presented in this paper.

2.2. Cooperation via a single inference rule

In the generic approach described above, the cooperation between systems is achieved at the upper layer of the OANTS architecture, by modelling each system as a separate inference rule. A more refined approach, first presented in [11], fosters cooperation by exploiting the lower layer of the OANTS blackboard architecture. This effectively cuts out the need to communicate via the central proof object, and therefore offers the advantage of a more succinct exchange of intermediate results between the systems involved, in a specialist language.

Direct bilateral integration of two reasoning systems is generally difficult if both systems do not share representation formalisms that are sufficiently similar. It also requires at least one of the systems involved to be open, such that it can continuously incorporate results of another system during its own run. Finally, the effort of implementing a dedicated inference rule for the communication of two particular types of reasoning systems is more involved than simply integrating a system and its results into OANTS's central architecture. It is therefore, necessary to carefully consider if a bilateral integration actually promises a gain in reasoning power of the overall system—see Section 4 for a detailed case study that empirically evaluates this specialist integration.

We realised the specialist approach discussed here by using a single inference rule to model the cooperation between the higher-order resolution prover LEO [8] and a first-order theorem prover in OANTS. The cooperation between the provers is enabled by directly exchanging sets of clauses without translating them into single formulae and back. The general idea is that LEO sends the subset of its clauses that do not contain any 'real' higher-order sub-terms (such as a λ-abstraction or embedded equations) to a first-order theorem prover. We call clauses of this sort *FO-like clauses*. In detail, the single inference rule 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 translating FO-like clauses directly is far more efficient than translating them first into single, fully quantified formulae in the central higher-order proof object. 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. Several first-order provers are integrated in OANTS and could be used, but in the case study that evaluates this approach (see Section 4) we used concretely BLIKSEM [17] and VAMPIRE [32]. In most cases, more than one process running a first-order ATP was necessary. This is because the subsets of FO-like clauses generated by LEO in its first reasoning loops are usually still consistent and they become inconsistent only after several reasoning rounds in which new FO-like clauses are generated. Each time the subset of FO-like clauses in LEO's search space changes, a new process running a first-order ATP is started. In contrast to the many processes running first-order ATPs only one process running LEO is started. Crucial to the success of the integration was also the possibility of retrieving intermediate results from LEO that are sufficiently informative for the first-order prover. 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, we had to forgo this optimisation and add all the clauses with primitive equality to the intermediate results. See [11] for more details on this aspect of the integration.

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. In fact, many theorems that would be difficult to prove or even not provable at all in reasonable time due to communication overhead in the generic architecture that uses a centralised communication, are now provable with this specialist architecture that decentralises communication—see the results of the case study in Section 4.3. A disadvantage of this approach is that we cannot easily translate and integrate the two proof objects produced by LEO and BLIKSEM, or LEO and VAMPIRE into the central proof object maintained by OANTS, as is possible when applying only one prover per open subgoal. The repercussions will be discussed in more detail in Section 4.2.

In the following two sections we will demonstrate the effectiveness of both integration approaches—the generic integration of arbitrary reasoning systems on the top layer of OANTS, and the specialist integration of a higher-order and a first-order resolution prover on the lower layer of OANTS—with example applications from diverse mathematical domains.

3. Generic integration of reasoning systems

In this section we present several application examples of the generic integration architecture as presented in Section 2.1. The examples demonstrate the ease of integration of systems in our architecture, the power of such cooperative architecture, and the diversity of examples that the combined systems can tackle.

In particular, we demonstrate the complementary interplay of theorem provers and model generators to decide validity of conditional and unconditional set-equalities as well as set-inequalities (Section 3.1.1). Our previous case studies further show the cooperation between an automated theorem prover and a computer algebra system on problems in sets over naturals (Section 3.1.2), and the collaboration of a higher-order and a first-order prover on group theory examples (Section 3.1.3). We then consider theoretical properties such as soundness and completeness for each case study (Section 3.2), and analyse their results (Section 3.3) to show that our novel cooperative solution achieves a much higher reliability and coverage than the standard single reasoning system approach.

3.1. Test problems

The test problems we chose for our three case studies are diverse and demonstrate the breadth and power of our cooperative reasoning approach. Pure set equalities have been previously investigated in [7]. Here in Section 3.1.1, we extend the case study to set inequalities, as well as set equalities and inequalities under additional conditions. The test problems in Section 3.1.2 on sets over naturals have in part previously appeared in [7]. The test problems in Section 3.1.3 in group theory have in part previously appeared in [36].

3.1.1. Conditional and unconditional set-equalities and set-inequalities

The examples in this section are concerned with checking the validity or invalidity of statements about set relations. This case study demonstrates the cooperation between higher-order ATP, first-order ATP and a model generator. The task at hand is to construct either a proof or a counterexample. Both are handled in a similar fashion by our system. First, simple natural deduction agents reduce the set equalities to a propositional logic statement. The resulting statements are then picked up by a propositional logic agent employing the theorem prover VAMPIRE and a counterexample agent using the model generator PARADOX. The logic statement is then either proved or disproved. Thus, valid and invalid statements are tackled analogously in all but the last step.

The test problems we consider are universally quantified statements involving equality, inequality, or the subset relations between sets constructed with the simple set operations $\cap, \cup, \setminus$ that represent intersection, union, and set-difference, respectively. Moreover, to widen the coverage to extreme ends of the spectrum of test problems, we consider additional conditions on the relationships between the sets involved. In particular, we explicitly express uniqueness or disjointness of particular sets.

As concrete examples of two set equalities whose validity/invalidity is to be decided, consider the proof or refutation of the following simple, unconditional statements:

$$\forall x, y, z_{\blacksquare}(x \cup y) \cap z = (x \cap z) \cup (y \cap z) \tag{1}$$

$$\forall x, y, z_{\blacksquare}(x \cup y) \cap z = (x \cup z) \cap (y \cup z) \tag{2}$$

Both problems are tackled similarly by the system. First, all the universally quantified variables are eliminated with the appropriate natural deduction rules. In the case of (1) this yields

$$(a \cup b) \cap c = (a \cap c) \cup (b \cap c)$$

as new open subgoal. Then set extensionality gives us

$$\forall u_{\blacksquare} u \in (a \cup b) \cap c \Leftrightarrow u \in ((a \cap c) \cup (b \cap c))$$

Further elimination of universally quantified variables and subsequent definition expansions of the operations $\cup, \cap$ and $\in$, with $a \cup b := \lambda z_{\blacksquare}(z \in a) \vee (z \in b)$, $a \cap b := \lambda z_{\blacksquare}(z \in a) \wedge (z \in b)$, and $u \in a := a(u)$ reduces that goal finally to

$$(a(d) \vee b(d)) \wedge c(d) = (a(d) \wedge c(d)) \vee (b(d) \wedge c(d))$$

which contains no variables and which is trivial to prove for any propositional logic prover. In case (2) we analogously derive

$$(a(d) \vee b(d)) \wedge c(d) = (a(d) \vee c(d)) \wedge (b(d) \vee c(d))$$

but now there exists a counterexample of the form $a(d), b(d), \neg c(d)$, which represents the set of all d such that $d \in a$, $d \in b$, but $d \notin c$. This counterexample can be easily constructed by a model generator.

3.1.2. Sets over naturals

The test problems in this case study consider sets over naturals, and demonstrate the cooperation between theorem provers and computer algebra systems. The problems themselves are mathematically trivial, but require a combination of deduction and computation to solve them. While the reasoning itself is relatively shallow, the problems can generally not be solved by a theorem prover alone as the contained functions need to be treated with symbolic or numerical computation that is out of reach of provers. In our architecture the problems are tackled by a collaboration of the higher-order theorem prover LEO, the first-order theorem prover OTTER, and the computer algebra system MAPLE.

A concrete example of these test problems is the following equation: $\{x \mid x > gcd(10, 8) \wedge x < lcm(10, 8)\} = \{x \mid x < 40\} \cap \{x \mid x > 2\}$. In order to tackle it, it is necessary to first simplify the numerical functions contained, that is gcd and lcm, and then rewrite the set representations on either side of the equality. The first step is carried out by a simplification agent which links the computer algebra system MAPLE to the core system. As an application condition, this agent checks whether the current subgoal contains certain simplifiable expressions. If so, it simplifies the subgoal by sending the simplifiable sub-terms (e.g., $x > gcd(10, 8)$) to MAPLE and replaces them with the corresponding simplified terms (e.g., $x > 2$). Hence, the new subgoal suggested by the simplification agent is, e.g.: $(\lambda x_{\blacksquare} x > 2 \wedge x < 40) = (\lambda x_{\blacksquare} x < 40) \cap (\lambda x_{\blacksquare} x > 2)$. Since no other agent comes up with a better alternative, this suggestion is selected and executed. Subsequently, the LEO agent successfully attacks the new goal after expanding the definition of $\cap$. This particular problem can, after MAPLE's contribution, be proved by LEO alone. In other cases MAPLE's contribution may lead to a problem which can be solved only by a further collaboration between LEO and a first-order ATP.

3.1.3. Group theory and algebra

The test problems in this case study involve theorems from group theory and algebra, and are mainly about the equivalence of definitions and uniqueness statements. This case study demonstrates the cooperation between higher-order and first-order ATPs. Since the problems contain some rather elaborate higher-order constructions, they cannot be tackled by a first-order theorem prover alone. Instead, they are solved by a goal directed higher-order natural deduction proof search in cooperation with a first-order automated theorem prover.

The group theory and algebra examples that we examined are rather easy from a mathematical viewpoint, however, can become non-trivial when painstakingly formalised. An example are proofs in which particular elements of one mathematical structure have to be identified by their properties and transferred to their appropriate counterparts in an enriched structure. The equivalence statement:

$$(\exists \circ_{\blacksquare} Group(G, \circ)) \Leftrightarrow (\exists \star_{\blacksquare} Monoid(M, \star) \wedge Inverses(M, \star, Unit(M, \star)))$$

where the unit element of the monoid has to be identified with the appropriate element of the group, is in this category. Here, *Group* and *Monoid* refer to a definition of a group and a monoid, respectively. $Inverses(M, \star, Unit(M, \star))$ is

a predicate stating that every element of M has an inverse element with respect to the operation $\star$ and the identity $Unit(M, \star)$. $Unit(M, \star)$ itself is a way to refer to that unique element of M that has the identity property.

In higher-order logic this can be formalised most elegantly using the description operator ι (cf. [1] for definition in higher-order logics), by assigning to the element in the group the unique element in the monoid that has exactly the same properties. In the context of our examples, we employed a description that encodes concepts like the (unique) unit element of a group by a single term that locally embodies the particular properties of the encoded concept itself. If any property of the unit element is required in a proof, then the description operator has to be unfolded (by applying a tactic in the system) and a uniqueness subproof has to be carried out.

The idea of the proofs is to divide the problems into smaller chunks that can be solved by automated theorem provers, and if necessary, to deal with formulae involving description. The ND search procedure implemented in OANTS has the task to simplify in turn the given formulae by expanding definitions and applying ND inferences. After each proof step the provers try to solve the introduced subproblems. If they all fail within the given resources (typically time limit), the system proceeds with the alternative ND inferences and subsequently the provers try to tackle the new subproblems introduced by them.

When a point is reached during the proof where neither applicable rules nor solutions from the provers are available, but the description operator still occurs in the considered problem, two theorems are applied to eliminate description. Description in goals is eliminated with the theorem $\forall Q_{\beta o}\forall P_{\beta o}[[\exists! x_{\beta} P(x)] \wedge [\forall z_{\beta} P(z) \Rightarrow Q(z)]] \Rightarrow [Q \iota P]$. Since the reverse direction of the above theorem does not necessarily hold (as with $Q\iota P$ we cannot assume that P actually uniquely describes an element), we use the theorem $\forall Q_{\beta o}\forall P_{\beta o}[Q\iota P] \Rightarrow [[\exists x_{\beta} P(x) \Rightarrow \forall y_{\beta} P(y) \Rightarrow (x = y)] \Rightarrow [\forall z_{\beta} P(z) \Rightarrow Q(z)]]$ to eliminate occurrences of description in forward reasoning direction. The results of description elimination are generally very large formulae, which can then again be tackled with the ND rules and the theorem provers.

3.2. Theoretical considerations

For the integration of multiple reasoning systems to construct a single, coherent proof, it is particularly crucial to consider the soundness of such an integration. In the case of combining different theorem provers (like in the case study in group theory examples in Section 3.1.3), this is mainly a question of compatibility and connectability of the respective calculi. We will shed more light on soundness and completeness considerations when presenting specialist integration of first-order and higher-order reasoners in Section 4.2. However, in the case of integrating more general computations into the reasoning process, for instance, as in our case with a computer algebra system or a model generator, soundness issues have to be considered case by case.

The use of a counterexample generator as presented in Section 3.1.1 is sound if and only if the full problem is given to the counterexample generator. Note that it is not sufficient to give to the counterexample generator only some of the axioms and not others, since this may transform an inconsistent set into a consistent one. In general the original problem is initially transformed into an equivalent problem that is given in parallel to a first-order theorem prover and a first-order model generator. The original problem is proved if the first-order theorem prover finds a refutation, and falsified if the first-order model generator finds a counterexample. Since the original transformation preserves (un-)satisfiability the approach is sound.

Soundness of the solutions for the problems in Section 3.1.2 depends crucially on soundness of the computations by the computer algebra system. This is not guaranteed, as MAPLE, like most other computer algebra systems, is not provably correct, and also does not provide an explicit justification for its computations. Consequently, correctness has to be ensured explicitly by certifying the computations after the proof has been found. This is achieved by constructing justifications on the calculus level—see [35] for details.

We cannot guarantee completeness neither in integrating a model generator nor a computer algebra system. A finite model generator is by its very nature incomplete. Similarly, a computer algebra system only provides a limited library of algorithms and generally has some limitations with respect to the size of its integer computations.

3.3. Results

3.3.1. Conditional and unconditional set-equalities and set-inequalities

We carried out a case study with an automatically and randomly generated test-bed of examples consisting of set statements involving equality, inequality and subset as relations, and $\cap, \cup, \setminus$ as functions that combine a maximum

Table 1
Summary of the results of the experiments with set expressions

Problem type	Valid	Invalid	Vampire			Paradox			Oants		
			Proof	Failed		Model	Failed		Proof	Model	Failed
Unconditional	**14**	**86**	**12**	**2**	**86**	**86**	**0**	**14**	**14**	**86**	**0**
$=$	2	38	0	2	38	38	0	2	2	38	0
$\subseteq$	12	33	12	0	33	33	0	12	12	33	0
$\neq$	0	15	0	0	15	15	0	0	0	15	0
Uniqueness	**20**	**80**	**3**	**17**	**80**	**13**	**67**	**20**	**20**	**80**	**0**
$=$	2	38	0	2	38	8	30	2	2	38	0
$\subseteq$	12	33	0	12	33	3	30	12	12	33	0
$\neq$	6	9	3	3	9	0	9	6	6	9	0
Disjointness	**31**	**69**	**6**	**25**	**69**	**69**	**0**	**31**	**31**	**69**	**0**
$=$	7	33	1	6	33	33	0	7	7	33	0
$\subseteq$	23	22	4	19	22	22	0	23	23	22	0
$\neq$	1	14	1	0	14	14	0	1	1	14	0

of 6 universally quantified variables up to a nesting depth of 6. For our experiments we generated 100 basic (i.e., unconditional) statements. All 100 statements are listed in Tables A.1 and A.2 in Appendix A. Given these 100 statements we then added conditions on the relationship between the universally quantified sets. In detail, we devised three types of formulae that can be viewed as the extreme positions of these conditions, by taking the set statement

(1) without assumptions, or *unconditional*, for example:

$$\forall x_1, x_2, x_3 . (x_1 \cup x_2) = (x_1 \cap (x_2 \cup x_3))$$

(2) under all different, that is, the *uniqueness* assumption, for example:

$$\forall x_1, x_2, x_3 . [(x_1 \neq x_2) \wedge (x_2 \neq x_3) \wedge (x_1 \neq x_3)] \rightarrow [(x_1 \cup x_2) = (x_1 \cap (x_2 \cup x_3))]$$

(3) under pairwise disjoint, or *disjointness*, assumption, for example:

$$\forall x_1, x_2, x_3 . [(x_1 \cap x_2 = \emptyset) \wedge (x_2 \cap x_3 = \emptyset) \wedge (x_1 \cap x_3 = \emptyset)] \rightarrow [(x_1 \cup x_2) = (x_1 \cap (x_2 \cup x_3))]$$

Note that there are formulae in the first class, such as $\forall x_1, x_2 . (x_1 \backslash x_2) \cup (x_2 \backslash x_1) \neq \emptyset$, which are non-theorems but are theorems in the second; or non-theorems in the first class such as $\forall x_1, x_2 . x_1 \backslash x_2 = x_1$, but theorems in the third.

Thus we carried out three sets of experiments, each with 100 problems. Each problem was then attempted to be proved by the theorem prover VAMPIRE, the model generator PARADOX, and OANTS separately within a two minutes time limit. If a system could not solve the problem within the given time, it was classified as a failed attempt. However, we distinguish between two types of failure, depending on the validity of the problem: VAMPIRE fails if it cannot prove a valid statement, whereas it is obviously not expected to solve an invalid statement. Conversely, PARADOX fails if it cannot find a model for an invalid statement. OANTS, on the other hand, is expected to solve all problems, thus there is only one type of failure possible.

A summary of the results is given in Table 1. It presents the number of valid and invalid set statements in each of the three experiments, followed by the number of problems VAMPIRE, PARADOX, and OANTS could or could not solve, respectively. Observe that for VAMPIRE and PARADOX figures for the two types of failure are given, separated by a dashed bar, where the first figure indicates the number of problems the system was indeed expected to solve. Each experiment is further broken down in the number of statements involving set equality, subset relation, and inequality. The full results for all three experiments are listed in Tables A.3, A.4, and A.5 in Appendix A.

The results in Table 1 present a mixed picture. In the case of unconditional statements the results are straightforward. PARADOX easily finds models for all invalid statements and VAMPIRE proves all valid statements except for the two involving equality. In the experiments with uniqueness condition, however, both PARADOX and VAMPIRE have considerably more problems. VAMPIRE cannot prove any statements with equality or subset relation, and only

proves 3 out of 6 of the inequality statements. Similarly, PARADOX fails to find models for the majority of statements with uniqueness condition, in particular it cannot find models for any of the invalid inequality statements. For statements with disjointness condition, PARADOX does better than VAMPIRE as it finds models for all invalid statements, whereas VAMPIRE can only prove a fraction of the valid statements.

In contrast, OANTS can solve all problems, regardless of precondition, and can also decide on their validity and invalidity, as it integrates both theorem proving and model generation. For the unconditional statements this is not as impressive, since using both, PARADOX and VAMPIRE, without integration one after another on all problems, is almost as powerful as OANTS. In fact, the timings in Table A.3 reveal that OANTS is slower due to communication overheads. On the other hand, for the conditional problems, OANTS is the only reliable way of distinguishing valid and invalid statements for all considered problems. Thus, the combined systems in OANTS have a considerably broader coverage of the problem domain than the single reasoning systems alone.

3.3.2. Sets over naturals

We experimented with 50 problems of sets over naturals. They all involved some element of computation, which was mainly concerned with natural number arithmetic, but also contained simple number theoretic functions such as *gcd*, *lcm*, and absolute value function. All of the problems could successfully be solved by the combination of first-order and higher-order reasoning and computer algebra simplification. As current automated theorem provers are comparatively weak in integer arithmetic and the formalisation of functions like *lcm* or *gcd* is very involved, they are likely to be outperformed by a combined system like OANTS, and hence, we did not carry out a comparison of OANTS with any specific automated theorem proving system.

3.3.3. Group theory and algebra

In our case study on group theory and algebra problems we have successfully experimented with 20 examples of the described type, of which we could solve 11 automatically. A direct comparison with first-order provers alone is superfluous, as in a purely first-order formulation quantified operations would have to be pre-instantiated appropriately. This would, however, defeat the purpose of showing equivalence of definitions that define different operations on a set.

There is still quite a number of problems which, while considered very easy for humans, cannot be solved automatically. As a simple example of a problem that could not be solved, consider proving the equivalence between two different definitions of a group. One definition is the standard one, that is, a group is a set G with an associative operation $\circ$, such that there exists a neutral element for $\circ$ and each element has an inverse with respect to that neutral element. The alternative, equivalent definition axiomatises a group as a set G with an operation $/$, such that for all $a, b, c \in G$ we have (i) $a/a = b/b$, (ii) $a/(b/b) = a$, (iii) $(a/a)/(b/c) = c/b$, and (iv) $(a/c)/(b/c) = a/b$. The main difficulty in the proof of this equivalence is to find the right reformulations between the two operations $\circ$ and $/$, since they cannot be simply equated. While in most of the other examples, higher-order unification can actually find the right equation between the operations in question, the complexity of the reformulation needed in this example would require a primitive substitution mechanism [3] that can handle the description operator. However, this is currently beyond automation.

3.3.4. Summary of results

Our integration approach has proved successful in different ways. Firstly, it provides a flexible means to integrate heterogeneous systems, especially those that are designed with diverse and even opposite functionality. The integrated system can solve tasks such as deciding the validity or invalidity of logical statements that can normally be done only by using several separate systems, or they cannot be proved at all with a single specialist system. Secondly, OANTS' range and coverage of solvable problems is far greater than that of any individual system alone. In particular, OANTS can formulate problems with the expressive power of a higher-order language, and is able to simplify problems in a goal-directed way either by using its own built-in calculus rules or by exploiting the computational power of other integrated systems (such as computer algebra) before handing them to dedicated first-order systems. And hence, thirdly, OANTS proves theorems in a significantly more reliable way than the individual systems.

While the approach described in this section is generic and allows to integrate several systems in a consistent and easy way, it leads to a computational overhead which in some cases makes proof search prohibitively expensive. In particular, experiments on integrating higher-order with first-order reasoners have shown that some problems quickly

reach sizes that make their translation via the OANTS central formalism computationally infeasible. This suggested a much tighter integration between component systems that cuts out overhead. In particular, it has led to the development of the specialist integration of reasoning systems for higher-order and first-order reasoners, which we describe and evaluate in the next section.

4. Specialist integration of reasoning systems

We now carry out a case study in which we empirically evaluate the power and efficiency of the specialist integration of reasoning systems described in Section 2.2. We are particularly interested in the higher-order/first-order ATP cooperation. Some of the results of this case study were previously reported in [11].

Existing higher-order ATPs generally exhibit shortcomings 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 [34], 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 have to be addressed. Nevertheless, for some mathematical problem domains, such as naive set theory, for instance, automated higher-order reasoning performs very well. Our motivation therefore is to combine the best of the two approaches in a specialist integration.

4.1. Test problems

We motivate the need for linking higher-order and first-order ATPs with some examples from Table 2. It contains a range of challenging problems taken from the TPTP [37], against which we will evaluate our system in Section 4.3. The selection is inspired by the one given in [21] but contains some additional problems.[3] The problems are given by the identifiers used in the SET domain of the TPTP-v3.0.1, 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 [2] 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 test problems in Table 2 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 2 are given in Table 3. Concepts are defined in terms of λ-expressions and they may contain other, already specified concepts. For presentation purposes, we use customary mathematical symbols ∪, ∩, 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 $A \cup B := \{x \mid x \in A \vee x \in B\}$. 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.

We present different example problems which demonstrate the main features of our specialist integration approach. We first discuss example SET171 + 3 to contrast our formalisation to a standard first-order one.

SET171+3 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 clause normalisation rules to reach some proper initial clauses. In our 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

[3] We omitted problem SET108 + 1 used in [21] as it addresses the universal class and can therefore not be formalised in type theory in the same concise way as the other examples.

Table 2
Test problems from TPTP for the evaluation of OANTS

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

and Bx is obtained from expansion of $x \in B$):

$$[(\lambda x_{\alpha_\blacksquare} Bx \vee (Cx \wedge Dx)) =^? (\lambda x_{\alpha_\blacksquare}(Bx \vee Cx) \wedge (Bx \vee Dx))]$$

Note that negated primitive equations are generally automatically converted by LEO into unification constraints. This is why $[(\lambda x_{\alpha_\blacksquare} Bx \vee (Cx \wedge Dx)) =^? (\lambda x_{\alpha_\blacksquare}(Bx \vee Cx) \wedge (Bx \vee Dx))]$ is generated, and not $[(\lambda x_{\alpha_\blacksquare} Bx \vee (Cx \wedge Dx)) = (\lambda x_{\alpha_\blacksquare}(Bx \vee Cx) \wedge (Bx \vee Dx))]^F$. 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

Table 3
Defined concepts occurring in test problems from Table 2

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

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 4).

We can observe that the assumptions provide only a partial axiomatisation of naive set theory. On the other hand, the specification introduces lemmas 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. It is therefore not surprising that most first-order ATPs fail to prove this

Table 4
TPTP problem SET171 + 3—distributivity of ∪ over ∩

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)

problem. Of course, the significance of the comparison is clearly limited, because different systems are optimised to a different degree. Moreover, first-order systems often use a case tailored problem representation (e.g., by avoiding some base axioms of the addressed theory), while the higher-order prover has a harder task of dealing with a general (not specifically tailored) representation that does not make any assumptions about which parts of the theory are or are not needed for the concrete problem at hand.

We use λ-abstraction as well as the extensionality treatment inherent in LEO's calculus [5]. 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 FO-like, 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):

$$[(\exists x_{\alpha_{\blacksquare}}(Bx \wedge (Cx \vee Dx)) \Leftrightarrow ((\exists x_{\alpha_{\blacksquare}} Bx \wedge Cx) \vee (\exists x_{\alpha_{\blacksquare}} Bx \wedge Dx))]^F$$

Normalisation results in 26 FO-like clauses, which present a hard problem for LEO: it needs approx. 35 seconds (see Section 4.3) 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, we get the following clause after definition expansion in problem SET646 + 3 (where $B_{o\alpha}, C_{o\alpha}, x_{\alpha}$ are again Skolem constants):

$$[Ax \Rightarrow (\forall y_{\beta_{\blacksquare}} By \Rightarrow (\forall u_{\alpha_{\blacksquare}} \forall v_{\beta_{\blacksquare}}(u = x \wedge v = y) \Rightarrow ((\neg\bot) \wedge (\neg\bot))))]^F$$

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:

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

which it normalises into:

$$[(\lambda x_{\alpha_{\blacksquare}}(Ax \wedge Bx)) =^? (\lambda x_{\alpha_{\blacksquare}} \bot)] \vee [(\lambda x_{\alpha_{\blacksquare}}(Ax \wedge \neg Bx)) =^? (\lambda x_{\alpha_{\blacksquare}} Ax)] \tag{9}$$

$$[(\lambda x_{\alpha_{\blacksquare}}(Ax \wedge Bx)) = (\lambda x_{\alpha_{\blacksquare}} \bot)]^T \vee [(\lambda x_{\alpha_{\blacksquare}}(Ax \wedge \neg Bx)) = (\lambda x_{\alpha_{\blacksquare}} Ax)]^T \tag{10}$$

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

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

$$[(\lambda x_{\alpha_\blacksquare}(Ax \wedge Bx)) = (\lambda x_{\alpha_\blacksquare}\bot)]^F \vee [(\lambda x_{\alpha_\blacksquare}(Ax \wedge \neg Bx)) = (\lambda x_{\alpha_\blacksquare}Ax)]^F \tag{11}$$

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 a first-order ATP.

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.

4.2. Theoretical considerations

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

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 [25] to be machine-checkable. Because of the tight integration of LEO with the first-order ATPs, in which FO-like clauses that are in LEO's search space are directly picked up by the processes running first-order ATPs, some essential information needed for TRAMP is not available. Thus the cooperative proof results of LEO-BLIKSEM or LEO-VAMPIRE cannot yet be translated and inserted into the centralised proof object and the generation of a machine-checkable proof object is not yet fully supported. One possible solution we propose is to translate the first-order proofs into LEO proofs and to insert them 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.

While there are many advantages in guaranteeing correctness of proofs by checking them, it is worth noting that the combination of LEO and BLIKSEM (or LEO and VAMPIRE) is sound under the assumption that the three 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 (or VAMPIRE) is inconsistent only if LEO generated an inconsistent set of clauses in the first place. By the assumption that BLIKSEM (or VAMPIRE) is sound follows that it 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 [23]. 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 (or VAMPIRE) 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 (or VAMPIRE). 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.

4.3. Results

We conducted several experiments to evaluate our cooperative reasoning approach. In particular, we concentrated on test problems given in Table 2. We investigated several LEO strategies in order to compare LEO's individual performance with the performance of the LEO-BLIKSEM and the LEO-VAMPIRE cooperation. Some of LEO's results were already published in [4,8], and some of LEO-BLIKSEM results were previously presented in [11]. We also compare our results to those of the most successful first-order systems on these test problems as given in the TPTP and in the literature. These were, in detail:

MUSCADET (v2.4) a natural deduction system that uses special inference rules for sets [31],
E-SETHEO (csp04) that combines a variety of first-order theorem provers and specialised decision procedures into a single proof engine exploiting strategy parallelism [28],
VAMPIRE (v7.0) a first-order theorem prover using a binary resolution and superposition based calculus [32],
SATURATE an extension of VAMPIRE with Boolean extensionality rules that are a one-to-one correspondence to LEO's rules for Extensional Higher-Order Paramodulation [21].

Table 5 presents the results of our experiments. The first column contains the TPTP identifier of the problem. The second column lists the TPTP 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 third, fourth, and fifth columns list whether SATURATE, MUSCADET and E-SETHEO, respectively, can (+) or cannot (–) solve a problem. The sixth column lists the timing results for VAMPIRE. All timings given in the table are in seconds. Since the results for SATURATE are taken from [21] (a '?' in Table 5 indicates that the result was not listed in [21] and is thus unavailable) and the results for MUSCADET and E-SETHEO are taken directly from the on-line version of the TPTP, a run-time comparison would be unfair as the times are measured on different platforms. The timings for VAMPIRE, on the other hand, are based on private communication with A. Voronkov. They were obtained on a computer with a very similar specification to the one we used, namely a 2.4 GHz Xenon machine with 1 GB of memory. The remaining three major columns: LEO, LEO-BLIKSEM, and LEO-VAMPIRE detail the results of our experiments. Each of these three columns is further divided into sub-columns to allow for a detailed comparison.

For our experiments with LEO alone in column seven in Table 5 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, EXT-INPUT-RECURSIVE (EIR). Column seven (LEO) in Table 5 presents the fastest strategy for each problem (Strat.), the number of clauses generated by LEO (Cl.), and the total run-time (Time). While occasionally there was 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 and LEO-VAMPIRE.[5] Each of column eight (LEO-BLIKSEM) and nine (LEO-VAMPIRE) in Table 5 presents the number of clauses generated by LEO (Cl.) together with the time (Time), and in addition, the number of first-order clauses (FOcl) sent to, the time (FOtm) used by, and the number of clauses generated (GnCl) by BLIKSEM and VAMPIRE, respectively.

The overall time limit of all experiments was 100 seconds for both LEO alone and the LEO-first-order cooperations. This time also includes the time needed to write and process input and output files over the network. While LEO and instances of BLIKSEM or VAMPIRE were running in separate threads (each run of BLIKSEM and VAMPIRE was given 50 seconds), the figures given in the 'Time' column reflect the overall time needed for a successful proof. That is, they contain the time needed by all concurrent processes: LEO's own process as well as those processes administering the various instances of BLIKSEM or VAMPIRE. However, during I/O operations (i.e., writing input files and reading output files of the provers) the threads were locked and could only be killed once the operation was completed. While these operations generally only took milliseconds, for particular large files this could take longer, which accounts for

[5] In a small empirical study we identified EXT-INPUT as the most successful strategy for LEO-BLIKSEM and LEO-VAMPIRE.

Table 5
Experimental data for the benchmark test problems given in Table 2

TPTP-Problem	Difficulty	Saturate	Mus cadet	E-Setheo	Vampire 7	LEO			LEO-BLIKSEM					LEO-VAMPIRE				
						Strat.	Cl.	Time	Cl.	Time	FOcl	FOtm	GnCl	Cl.	Time	FOcl	FOtm	GnCl
SET014 + 4	.67	+	+	+	.01	ST	41	.16	34	6.76	19	.01	7	11	2.6	.01	.01	16
SET017 + 1	.56	–	–	+	.03	EXT	3906	57.52	25	8.54	16	.01	74	28	5.05	8	.01	22
SET066 + 1	1.00	?	–	–	–	–	–	–	26	6.80	20	.01	56	38	3.73	17	.01	53
SET067 + 1	.56	+	+	+	.04	ST	6	.02	13	.32	16	.01	12	9	.1	10	.01	17
SET076 + 1	.67	+	–	+	.00	–	–	–	10	.47	18	.01	35	12	.97	12	.01	27
SET086 + 1	.22	+	–	+	.04	ST	2	.01	2	.01	N/A	N/A	N/A	2	.01	N/A	N/A	N/A
SET096 + 1	.56	+	–	+	.03	–	–	–	27	7.99	14	.01	25	81	7.29	71	0.02	23
SET143 + 3	.67	+	+	+	68.71	EIR	37	.38	33	7.93	18	.01	19	8	.31	9	.01	9
SET171 + 3	.67	+	+	–	108.31	EIR	36	.56	25	4.75	19	.01	20	6	.38	10	.01	9
SET580 + 3	.44	+	+	+	14.71	EIR	25	.19	6	2.73	8	.01	13	8	.23	12	.01	4
SET601 + 3	.22	+	+	+	168.40	EIR	145	2.20	55	4.96	8	.01	13	20	1.18	31	.01	17
SET606 + 3	.78	+	–	+	62.02	EIR	21	.33	17	10.8	15	.01	5	5	.27	5	.01	3
SET607 + 3	.67	+	+	+	65.57	EIR	22	.31	17	7.79	15	.01	6	5	.26	8	.01	3
SET609 + 3	.89	+	+	–	161.78	EIR	37	.60	26	6.50	19	.01	17	6	.49	10	.01	9
SET611 + 3	.44	+	–	+	60.20	EIR	996	12.69	72	32.14	38	.01	101	39	4.00	40	0.03	23
SET612 + 3	.89	+	–	–	113.33	EIR	41	.54	18	3.95	6	.01	7	8	.46	11	.01	9
SET614 + 3	.67	+	+	–	157.88	EIR	38	.46	19	4.34	16	.01	17	8	.41	9	.01	9
SET615 + 3	.67	+	+	–	109.01	EIR	38	.57	17	3.59	6	.01	9	6	.47	8	.01	9
SET623 + 3	1.00	?	–	–	–	EXT	43	8.84	23	9.54	10	.01	14	9	2.27	10	.01	8
SET624 + 3	.67	+	–	+	.04	ST	4942	34.71	54	9.61	46	.01	212	47	3.29	44	.01	71
SET630 + 3	.44	+	–	+	60.39	EIR	11	.07	6	.08	8	.01	4	4	.05	6	.01	10
SET640 + 3	.22	+	–	+	70.41	EIR	2	.01	2	.01	N/A	N/A	N/A	2	.01	N/A	N/A	N/A
SET646 + 3	.56	+	–	+	59.63	EIR	2	.01	2	.01	N/A	N/A	N/A	2	.01	N/A	N/A	N/A
SET647 + 3	.56	+	–	+	64.21	EIR	26	.15	13	.30	13	.01	15	7	.12	7	.01	11
SET648 + 3	.56	+	–	+	64.22	EIR	26	.15	14	.30	13	.01	16	7	.12	9	.01	3
SET649 + 3	.33	–	–	+	63.77	EIR	45	.30	29	5.49	12	.01	16	10	.25	13	.01	8
SET651 + 3	.44	–	–	+	63.88	EIR	20	.10	11	.16	10	.01	11	7	.09	8	.01	2
SET657 + 3	.67	+	–	+	1.44	EIR	2	.01	2	.01	N/A	N/A	N/A	2	.01	N/A	N/A	N/A
SET669 + 3	.22	–	–	+	.34	EI	6	.19	7	.21	N/A	N/A	N/A	6	.2	N/A	N/A	N/A
SET670 + 3	1.00	?	–	–	–	EXT	15	.17	17	.36	16	.01	6	9	.14	11	.01	14
SET671 + 3	.78	–	–	+	218.02	EIR	78	.64	7	2.71	10	.01	14	13	.47	11	.01	9
SET672 + 3	1.00	?	–	–	–	EXT	27	.4	30	.70	21	.01	11	10	.23	12	.01	14
SET673 + 3	.78	–	–	+	47.86	EIR	78	.65	14	5.66	14	.01	16	13	.47	17	.01	6
SET680 + 3	.33	+	–	+	.07	ST	185	.88	29	4.61	18	.01	24	30	2.38	16	.01	27
SET683 + 3	.22	+	–	+	.06	ST	46	.20	35	8.90	18	.01	24	12	.27	15	.01	4
SET684 + 3	.78	–	–	+	.33	ST	275	2.45	46	5.95	26	.01	47	41	3.39	35	.01	38
SET686 + 3	.56	–	–	+	.11	ST	274	2.36	46	5.37	26	.01	46	42	3.55	37	.01	39
SET716 + 4	.89	+	+	–	–	ST	39	.45	18	3.81	18	.01	118	19	.4	24	0.02	73
SET724 + 4	.89	+	+	–	–	EXT	154	2.75	18	7.21	15	.01	23	10	1.91	14	.01	20
SET741 + 4	0.91	?	+	–	–	–	–	–	21	92.76	22	.01	104 850	21	3.70	26	.01	570
SET747 + 4	.89	–	+	–	–	ST	34	.46	25	1.11	18	.01	10	11	1.18	8	.01	14
SET752 + 4	.89	?	+	–	–	–	–	–	50	6.60	48	.01	4363	50	516.0	48	.01	4 145 104
SET753 + 4	.89	?	+	–	–	–	–	–	15	3.07	12	.01	19	12	1.64	12	.01	47
SET764 + 4	.56	+	+	+	.02	EI	2	.01	2	.01	N/A	N/A	N/A	2	.01	N/A	N/A	N/A
SET770 + 4	.89	+	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

the more than 500 s needed by LEO-VAMPIRE on problem SET752+4. Since all processes ran on a single processor, there is potential to improve upon run times by using real multiprocessing. Note also, that the number of clauses in LEO's search space is typically low as 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 or VAMPIRE) than there are in LEO's search space. Finally, observe that some problems were refuted after LEO's clausal normalisation, and hence BLIKSEM or VAMPIRE was not applicable (N/A).

The results in columns three through six indicate that some problems are still very hard for first-order ATPs, as well as for the special purpose theorem prover MUSCADET. While LEO itself can solve a majority of these problems

with some strategy, the LEO-BLIKSEM and LEO-VAMPIRE cooperations can solve more problems and, moreover, need only a single LEO strategy. We can also observe that many problems that appear to be relatively hard for LEO alone (e.g., SET017 + 1, SET611 + 3, SET624 + 3), are not only more quickly solved in the cooperative approach, but are often reduced to a relatively small higher-order preprocessing step with subsequent easy first-order proofs, as for instance, in the case of SET017 + 1. When comparing LEO-BLIKSEM and LEO-VAMPIRE one can observe that neither is clearly better than the other. Both cooperations solve the respective sub-problems very quickly, and it depends on the problem as to which one finds a solution earlier in the proof search. Moreover, the number of clauses produced during proof search for BLIKSEM and VAMPIRE can vary significantly. This suggests that in general, it is advantageous to integrate more than one first-order prover into the cooperative multi-agent architecture.

From a mathematical viewpoint the investigated problems are easy and, hence, they should ideally be reliably and very efficiently solvable within a proof assistant. This has been achieved for the examples in Table 5 (except for SET770 + 4) by our cooperative 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

There exist several systems that are related to ours from the point of view of integrating a number of reasoners in a cooperative environment. The main difference is that none of these integrate such a variety of diverse reasoning systems in a common cooperative and distributed framework. Here we outline commonalities and differences of some such systems with our work.

The integration of reasoners and reasoning strategies was pioneered in the TEAMWORK system [19], which realises the cooperation of different reasoning strategies, and the TECHS system [18], which realises a cooperation between a set of heterogeneous first-order theorem provers. Similar 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 [20]) is that our system bridges not only between first-order theorem provers, but also between first-order ATPs, higher-order ATPs, computer algebra systems and model generators. 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 [22] which realises a generic interface between HOL and first-order theorem provers. It is similar to the solution previously achieved by TRAMP [25] 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 and first-order ATPs is presented in [26,27]. The main difference of our work to the related systems is that while our system calls first-order provers from within automatic higher-order proof search, this is not the case for [22,25–27].

More generally, our cooperative, distributed and multi-agent framework is related to work on parallelism of deduction and other multi-agent architectures. The notion of parallelism in deduction has been categorised into three types [13]: parallelism on term level, on clause level and on search level. Our architecture models all three types. Parallelism on term level is clearly realised since our agents can access sub-terms in parallel during their search (for details of the OANTS suggestion mechanism, see [10]). Parallelism on clause level is mainly concerned with term rewriting steps, which corresponds loosely to the application of tactics in our system, and this is already realised in our underlying OMEGA system. Parallelism on search level is one of the key criteria of our system, since in each step the search for the next applicable proof step, which is either a single tactic or the computations of an integrated reasoner, is parallelised.

Multi-agent aspects of our framework can be compared with respect to the characterisation of agents as autonomous and flexible computational entities that exhibit, to a varying degree, reactive, pro-active and social abilities [39,40]. Our agents are clearly autonomous from a software engineering point of view, since they are implemented in concurrent threads. They are pro-active (i.e., they are not scheduled), reactive (i.e., their internal state is used to react to changes in the overall system) and robust. They cooperate, but do not exhibit any real social abilities, that is, they are not aware of other agents and individually cannot decide autonomously whom to cooperate with. They also have no real planning capabilities. From these points of view, our system is more closely related to distributed problem solving

systems like HASP [30] and POLIGON [33], rather than advanced layered agent architectures like INTERRAP [29] and the one by Bond and Gasser [14].

6. Conclusion

We presented a general agent-based architecture that enables easy integration and cooperation of diverse and very heterogeneous systems like higher-order, first-order automated theorem provers, computer algebra systems and model generators. We showed how the integration of different reasoning systems can lead to a combined system which goes significantly beyond the capabilities of each individual system. For this we experimented with different forms of integration which can be achieved relatively easily in the OANTS-framework. OANTS provides a framework which enables a generic integration of a variety of systems, and is based on the construction of a common proof object. The results of our case studies in Section 3 demonstrate a stronger reliability and coverage of our system in comparison to other single specialist systems.

In the case of some specialist reasoning systems it pays off to integrate the different systems more tightly so that they can communicate more efficiently in a specialist language. The results of this case study in Section 4 provide evidence and support this, in particular they show the effectiveness of a specialist integration between a higher-order and first-order resolution theorem prover. Our non-optimised system outperforms state-of-the-art first-order theorem provers and their ad hoc extensions such as SATURATE [21] 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* lemmas 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 [22,26, 27]) that support our hybrid reasoning approach.[6] They also provide further evidence for the usefulness of the OANTS approach as described in [7,10] for flexibly modelling the cooperation of reasoning systems. The results also indicate 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 can be evaluated against higher-order systems (and vice versa)—such an extension of the TPTP library to higher-order logic is currently under way.

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 lemmas 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 [12] and Brown's PhD thesis on *set comprehension in Church's type theory* [15].

Appendix A. Testing the generic integration

The following tables contain the data relevant to the experiments on set statements described in Section 3. Tables A.1 and A.2 contain the problem formulations of the unconditional statements. Tables A.3, A.4 and A.5 contain the detailed results of the experiments with VAMPIRE, PARADOX, and OANTS applied to the statements without conditions, with uniqueness condition and with disjointness condition, respectively. All experiments were run on a PC with four 2.4 GHz Xeon CPU and 4 GB of memory running Linux.

In Tables A.3–A.5 all timings are given in seconds, where t/o means time out after 120 seconds and $< .1$ means that the time was below the measurability of the operating system. For problems that the system was not expected to solve, that is, invalid problems for VAMPIRE or valid problems for PARADOX, the table will always have a t/o entry although the systems occasionally detected the invalidity or validity, respectively. The VAMPIRE column contains the number of clauses generated during proof search. The PARADOX column contains the minimal size of model needed to refute a conjecture. Finally, the OANTS column gives first the nature of the result, that is, proof or counterexample

[6] Indeed, we are currently investigating in a new project LEO-II a cooperation between a re-implementation of the LEO prover and tightly integrated first-order theorem provers.

Table A.1
Automatically generated set equality test problems for the evaluation of OANTS

#	Formalisation	
1	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((e \setminus d) \setminus e) \cap ((f \cup c) \cup (d \cup f))) \cap (((d \cap b) \setminus (f \setminus b)) \setminus (c \setminus d))) \subseteq (c \cap (((d \cap a) \cap c) \cup ((f \cap d) \cup (d \cup a))))$
2	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((e \cup f) \cap (d \setminus e)) \cup (d \cap a)) \setminus ((f \cup (d \cup b)) \setminus ((f \cap a) \setminus c))) = e$
3	$\forall a,b,c,d,e,f_{\blacksquare}$	$(b \cap (((f \cap d) \setminus (b \cap f)) \cap b)) = (f \setminus (c \setminus ((a \cup e) \cup (c \setminus b))))$
4	$\forall a,b,d,e,f_{\blacksquare}$	$((((f \setminus d) \cap (b \cap d)) \setminus d) \cup a) = ((((a \cup f) \cap (b \cap e)) \cup b) \cap (((d \cap a) \setminus (b \setminus d)) \cup a))$
5	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((d \cap a) \cap a) \cap a) \cap (d \cap ((c \setminus b) \setminus (c \cup e)))) \subseteq ((((b \cup d) \setminus (f \cap e)) \cap ((f \cap b) \cap (a \cup b))) \cap c)$
6	$\forall a,b,c,d,e,f_{\blacksquare}$	$a = ((((e \cap a) \cap c) \cap ((b \cup a) \cap f)) \setminus (((e \cup c) \cap (c \cap d)) \setminus ((a \cap d) \cap b)))$
7	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((d \cap c) \cup (a \cap c)) \cup e) \cap f) = (b \cap d)$
8	$\forall a,b,c,d,e_{\blacksquare}$	$((((e \cap c) \cup (c \setminus d)) \cup (a \setminus (c \setminus b))) \setminus ((b \cup (d \cup a)) \setminus c)) = (((d \cap (e \cup b)) \cap ((d \setminus c) \setminus (d \cap b))) \setminus (e \cup b))$
9	$\forall b,c,d,e,f_{\blacksquare}$	$(e \setminus (f \cup ((d \cup f) \cup b))) \subseteq (((e \cap (e \cup f)) \setminus (c \cap (e \cap f))) \setminus d)$
10	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((c \cup e) \cup ((d \setminus b) \cap f)) \cup (((d \setminus b) \setminus (a \cup f)) \setminus ((a \cap f) \setminus (d \cap f)))) = b$
11	$\forall a,b,d,e,f_{\blacksquare}$	$((b \setminus (f \cup (b \setminus e))) \cup d) \subseteq a$
12	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((c \setminus d) \cup (d \cap f)) \cap ((c \cap a) \setminus d)) \cup (((d \cup a) \setminus (d \setminus e)) \setminus a)) \subseteq$ $((f \setminus ((e \cap c) \cap (f \cup b))) \cap (((e \cap b) \cap a) \setminus ((a \setminus b) \cup (b \cup f))))$
13	$\forall a,c,d,e,f_{\blacksquare}$	$(((f \setminus (e \setminus a)) \cap (c \setminus (e \setminus d))) \cap (((c \setminus d) \cap a) \cap e)) = ((a \setminus (c \cap (d \setminus e))) \cap f)$
14	$\forall a,b,c,d,e,f_{\blacksquare}$	$((b \cap e) \setminus (((e \cup d) \cup e) \cup a)) = ((((e \setminus a) \cap f) \cap ((d \setminus b) \cap (b \cap d))) \setminus (((f \cup a) \cup (c \cup b)) \cap ((c \setminus f) \cap (e \cup d))))$
15	$\forall a,b,c,d,f_{\blacksquare}$	$c \subseteq ((((a \setminus b) \setminus d) \setminus ((a \cup c) \setminus (f \cap b))) \cup (d \setminus ((b \cap c) \setminus (d \setminus f))))$
16	$\forall b,c,d,e,f_{\blacksquare}$	$(d \cup (e \cap (f \setminus (e \cup d)))) = (((d \cup (f \setminus c)) \setminus b) \setminus ((f \cup (f \setminus b)) \cup ((b \cup c) \cup (b \cup f))))$
17	$\forall b,c,d,e,f_{\blacksquare}$	$b \subseteq ((((f \setminus b) \cup (f \cap d)) \cup c) \cup (((b \setminus e) \cup (e \cup b)) \setminus b))$
18	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((f \setminus a) \cap f) \setminus e) \cap (((a \setminus d) \cup (c \setminus a)) \cap ((f \setminus c) \cap (b \setminus d)))) \subseteq ((((f \cup d) \cap (a \cap e)) \cap (d \cup (d \cup b))) \cap f)$
19	$\forall a,b,c,d_{\blacksquare}$	$(a \cup ((c \cup (b \cap c)) \cap ((d \setminus b) \cap (c \cap d)))) = c$
20	$\forall a,b,c,d,e,f_{\blacksquare}$	$(f \setminus (((a \setminus d) \cap (e \setminus a)) \setminus c)) \subseteq ((((b \setminus a) \cup (b \setminus e)) \cup a) \cup b)$
21	$\forall a,b,c,e,f_{\blacksquare}$	$(f \cap (((c \cup f) \cap a) \cap ((b \cup e) \cap (c \cup e)))) \subseteq c$
22	$\forall a,b,c,d,e,f_{\blacksquare}$	$a \neq ((((e \cap a) \cap c) \cap ((b \cup a) \cap f)) \setminus (((e \cup c) \cap (c \cap d)) \setminus ((a \cap d) \cap b)))$
23	$\forall a,b_{\blacksquare}$	$((a \setminus b) \cup (b \setminus a)) \neq \emptyset$
24	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((a \cap c) \cap (d \cup c)) \cup ((e \cup a) \cup d)) \cap (((f \cap e) \setminus (e \setminus b)) \cup ((a \cap f) \cup d))) =$ $((b \cup (c \setminus (e \setminus c))) \setminus (((a \setminus f) \cup c) \cup ((a \cup b) \cap e)))$
25	$\forall a,b,c,d,e,f_{\blacksquare}$	$((a \cap (e \cup (f \cap b))) \cup (((b \cap e) \setminus b) \cap ((f \cap b) \setminus d))) \subseteq (d \cap ((a \cap (c \cap e)) \setminus ((a \cup e) \setminus d)))$
26	$\forall a,b,c,d,e,f_{\blacksquare}$	$(c \cup (((b \setminus c) \cup (b \setminus d)) \cap c)) = (b \cap (((c \setminus a) \cap (f \cup e)) \cap d))$
27	$\forall a,c,d,e_{\blacksquare}$	$e \subseteq ((((e \cup a) \setminus (c \setminus a)) \cup c) \cap (a \setminus d))$
28	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((a \cup f) \setminus f) \cup ((a \cap e) \cap (a \setminus f))) \setminus (d \cup b)) =$ $((((b \cup d) \setminus (a \cup d)) \cup ((a \cup d) \setminus f)) \cap ((c \setminus (d \setminus f)) \cap ((e \cap d) \cap (f \setminus d))))$
29	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((f \cap c) \cap ((c \cap a) \cap (a \setminus c))) \cap e) \subseteq (((b \cup (b \setminus d)) \setminus ((f \cup e) \setminus e)) \cup (d \setminus e))$
30	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((e \cup (e \cup d)) \setminus ((b \setminus e) \cap a)) \cap (a \cap f)) \subseteq ((e \setminus c) \setminus ((((e \cup d) \setminus b) \cup ((e \setminus a) \cap (d \cap a)))))$
31	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((d \cap c) \cup (a \cap c)) \cup e) \cap f) \neq (b \cap d)$
32	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((a \setminus b) \cup (b \setminus c)) \cup (((c \setminus d) \cup (d \setminus e)) \cup ((f \setminus a) \cup (e \setminus f)))) \neq \emptyset$
33	$\forall b,c,d,e,f_{\blacksquare}$	$((((f \cup b) \cap (e \cup c)) \cup ((f \setminus d) \setminus c)) \cup ((e \cap (c \cap f)) \cup ((d \cap b) \cup e))) \subseteq ((((b \setminus e) \cap e) \cap ((b \setminus c) \setminus (f \setminus b))) \cup c)$
34	$\forall a,b,c,d,e,f_{\blacksquare}$	$(d \cup (((b \cap f) \setminus (e \cap b)) \cup (d \setminus f))) = (b \setminus (((e \cap c) \cap (d \cap f)) \setminus ((f \cap b) \setminus (a \cup f))))$
35	$\forall a,b,d,e,f_{\blacksquare}$	$((((b \cup a) \cap e) \cap ((f \cup d) \cap (f \setminus b))) \cup d) = a$
36	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((f \setminus a) \cap f) \cap (f \setminus (d \setminus e))) \cap (((f \cup b) \setminus (e \setminus f)) \cup ((c \cap b) \setminus (b \cup c)))) \subseteq$ $((a \cup (e \setminus a)) \cap (((e \cup b) \setminus (c \setminus f)) \setminus (f \setminus c)))$
37	$\forall a,c,d,e,f_{\blacksquare}$	$((((d \cup c) \setminus (c \cap a)) \cup ((f \cup \emptyset) \cap (a \setminus e))) \cap c) \neq a$
38	$\forall a,b,c,d,e,f_{\blacksquare}$	$f = ((((d \setminus e) \setminus (b \setminus c)) \cup ((f \setminus c) \cap (d \setminus c))) \setminus a)$
39	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((b \cap e) \setminus (c \cap a)) \cup b) \cap (((c \cap a) \setminus (d \cap f)) \cup ((a \cap f) \cup (a \setminus c)))) \subseteq$ $((((f \cup a) \cup (d \cap e)) \setminus (a \cup (f \cap e))) \setminus (((d \cup e) \cup (f \cup d)) \cap ((a \cup d) \setminus (f \cap e))))$
40	$\forall a,b,c,d,e,f_{\blacksquare}$	$(a \setminus (((f \setminus d) \cap (b \setminus d)) \setminus c)) \subseteq ((((b \cap d) \cup (c \cap d)) \setminus ((e \cap c) \cup (a \cap d))) \setminus (((d \cap e) \setminus (f \cap d)) \cup (d \cap (e \cup b))))$
41	$\forall a,b,c,d,e,f_{\blacksquare}$	$(e \cap (((e \setminus a) \setminus (f \cap c)) \setminus ((a \cap e) \setminus (c \setminus a)))) = (c \cup (((c \cap f) \cap (a \setminus b)) \setminus ((b \cap d) \cup (c \cup a))))$
42	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((((a \setminus b) \cup c) \setminus a) \setminus c) \setminus f) \neq (((a \setminus b) \cup (b \setminus c)) \cup ((c \setminus d) \cup ((d \setminus e) \cup ((f \setminus a) \cup ((e \setminus f) \cup (f \setminus c))))))$
43	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((e \cap b) \cup e) \cap (f \cap (a \cap c))) \cup (((d \cup e) \cap (d \setminus c)) \cup d)) =$ $((((b \cap a) \cap (f \cup d)) \setminus b) \setminus (((b \setminus e) \cup (e \cup a)) \setminus ((e \setminus b) \cup (e \setminus f))))$
44	$\forall a,b,c,d,e,f_{\blacksquare}$	$(b \setminus e) = ((((e \setminus d) \setminus (d \cap a)) \setminus ((e \cup c) \cup (f \cup d))) \cup (((f \setminus (e \cup c)) \cup ((a \cap e) \cup (b \cup e)))))$
45	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((c \cup a) \cap (f \cup e)) \cup (e \cap (f \cup b))) \cap (((b \cap e) \cup b) \setminus ((a \cap b) \setminus (d \setminus a)))) =$ $((b \setminus c) \setminus (((d \cup e) \setminus (f \setminus c)) \cap ((d \cap a) \cup (e \cap f))))$
46	$\forall a,b,c,d,e_{\blacksquare}$	$(e \cap b) = ((((c \cup a) \cap (b \setminus a)) \setminus e) \cap ((a \setminus c) \cap ((d \setminus e) \cap e)))$
47	$\forall a,b,c,d,e_{\blacksquare}$	$((((d \cap a) \cup (d \cup b)) \setminus e) \cup (a \cap ((e \setminus b) \setminus (e \setminus c)))) =$ $((((d \cup a) \setminus (d \cap c)) \cup ((e \cup d) \cap (b \setminus c))) \cup (((a \cap b) \setminus b) \setminus (a \cup (a \cup c))))$
48	$\forall a,b,c,d,e,f_{\blacksquare}$	$(d \setminus (((a \cap d) \cup f) \cap f)) \subseteq (((d \cap (c \setminus d)) \cup (b \cap (b \setminus c))) \setminus (f \cap ((e \setminus b) \setminus (b \cup c))))$
49	$\forall a,c,d,e,f_{\blacksquare}$	$f = ((((e \setminus c) \cup (f \setminus a)) \cup ((d \cap e) \cap (a \cup c))) \setminus f)$

Table A.2
Automatically generated set equalities continued

#	Formalisation	
50	$\forall a,b,c,d,e,f_{\blacksquare}$	$((d \cap (c \cup (d \setminus f))) \cup b) \subseteq ((((f \setminus e) \cap (e \setminus a)) \cup ((e \setminus f) \cup e)) \cup d)$
51	$\forall a,b,c,d,e,f_{\blacksquare}$	$(b \cap e) = (a \cup (((e \cup f) \cap (c \setminus d)) \cup ((d \setminus e) \cup d)))$
52	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((b \setminus d) \setminus (a \cup c)) \cup ((e \cap a) \cap (a \cap c))) \cup (f \cap ((d \cup c) \cap e))) = ((e \cup ((e \cap f) \cap (c \cap e))) \cup (b \cap (d \setminus (e \cup f))))$
53	$\forall a,b,c,d,e,f_{\blacksquare}$	$((f \cap ((a \setminus e) \setminus (f \setminus b))) \cap (((b \setminus d) \setminus d) \cup ((d \cup a) \cap (c \cap b)))) \subseteq (d \cup (c \cup (d \cap f)))$
54	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((d \cap e) \cap (d \setminus f)) \cup (d \cap (a \cup e))) \setminus (((e \setminus b) \cap e) \setminus ((b \setminus f) \cap (b \setminus e)))) \subseteq ((a \cup b) \setminus (b \cap ((c \cup b) \setminus (a \setminus c))))$
55	$\forall a,b,c,d,e_{\blacksquare}$	$((((e \cap c) \cup (c \setminus d)) \cup (a \setminus (c \setminus b))) \setminus ((b \cup (d \cup a)) \setminus c)) \neq (((d \cap (e \cup b)) \cap ((d \setminus c) \setminus (d \cap b))) \setminus (e \cup b))$
56	$\forall a,b,d,e,f_{\blacksquare}$	$((((a \setminus e) \cap (d \setminus e)) \cap ((e \setminus a) \cup (a \cap b))) \setminus b) \subseteq (d \cup ((f \setminus d) \cup (a \setminus (d \cup a))))$
57	$\forall a,b,c,d,e,f_{\blacksquare}$	$b \subseteq ((((e \cap c) \cup (d \setminus e)) \cap ((b \setminus a) \setminus d)) \setminus (((f \cup c) \setminus (c \setminus b)) \cap ((f \setminus e) \cup (d \cup c))))$
58	$\forall b,c,d,e,f_{\blacksquare}$	$(e \setminus (f \cup ((d \cup f) \cup b))) \neq (((e \cap (e \cup f)) \setminus (c \cap (e \cap f))) \setminus d)$
59	$\forall a,b,c,d,e,f_{\blacksquare}$	$((b \cup ((a \cup f) \setminus (a \cap d))) \setminus (e \setminus b)) \subseteq ((b \setminus ((f \cap d) \cap (d \cup f))) \setminus (c \setminus ((d \cup f) \cap (b \setminus f))))$
60	$\forall b,c,d,e_{\blacksquare}$	$(b \cap (c \cap ((d \cup e) \setminus (d \setminus c)))) = c$
61	$\forall a,b,d,e_{\blacksquare}$	$(((e \cup (e \setminus b)) \cup e) \setminus ((e \cap (b \cap e)) \cup ((d \cup a) \cup d))) = d$
62	$\forall a,b,c,f_{\blacksquare}$	$a = (b \setminus (((c \setminus f) \cap (b \cup c)) \cup a))$
63	$\forall a,b,c,e,f_{\blacksquare}$	$(e \cap ((a \cap (f \cap a)) \cup ((a \cap e) \setminus (c \cup b)))) \subseteq (a \cup (e \cap b))$
64	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((b \cap f) \cap (e \cup a)) \cup ((a \setminus f) \cup (d \cap e))) \cap (((a \cap f) \setminus (a \cup f)) \cap ((f \cup d) \setminus (a \cup e)))) = ((((c \cup f) \cup (c \cap e)) \cup ((b \cup c) \cap (b \cup d))) \cap (((c \setminus e) \cap b) \cap e))$
65	$\forall a,b,c_{\blacksquare}$	$(((a \setminus b) \cup (b \setminus c)) \cup (c \setminus a)) \neq \emptyset$
66	$\forall a,b,d,f_{\blacksquare}$	$((((a \setminus d) \cup (b \cap a)) \cup (d \setminus (a \cap b))) \setminus (((d \cap b) \cup a) \cup a)) = f$
67	$\forall a,b,c,e,f_{\blacksquare}$	$c \subseteq (b \setminus (((c \setminus f) \cap (c \cup a)) \setminus e))$
68	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((c \cup e) \cup ((d \setminus b) \cap f)) \cup (((d \setminus b) \setminus (a \cup f)) \setminus ((a \cap f) \setminus (d \cap f)))) \neq b$
69	$\forall a,b,c,e,f_{\blacksquare}$	$((((e \cap c) \setminus (b \setminus a)) \cup b) \cap (((c \setminus a) \cup b) \setminus ((a \cup f) \cap b))) \subseteq (((e \cap (b \cup c)) \setminus ((f \cap e) \cup (e \cup c))) \setminus (f \cup a))$
70	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((e \setminus f) \setminus (d \cup c)) \setminus (a \setminus (d \cup a))) \cup b) \subseteq (d \cap (a \cap (a \cup (a \setminus c))))$
71	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((f \cap e) \cup b) \setminus ((d \setminus f) \cap (a \cap f))) \setminus ((b \cap (\emptyset \cap d)) \cap (\emptyset \cap (a \cup e)))) \neq ((((f \setminus d) \setminus (d \cap c)) \cap c) \cap (((c \cup e) \cup c) \cap e))$
72	$\forall a,c,d,e,f_{\blacksquare}$	$(((c \setminus (a \cap d)) \cup ((c \setminus e) \setminus (c \setminus d))) \setminus ((c \setminus (d \setminus c)) \cup ((d \setminus a) \setminus c))) = f$
73	$\forall a,b,c,d,e,f_{\blacksquare}$	$c = (((c \cup (a \cup e)) \cup c) \cup ((d \cup (a \cup f)) \setminus ((b \cup c) \setminus a)))$
74	$\forall c,d,e,f_{\blacksquare}$	$((f \cap ((f \setminus e) \setminus e)) \cap ((e \cap c) \cup ((c \cup e) \cup (e \cap c)))) = (d \setminus (f \cap ((c \cup e) \setminus (e \cup f))))$
75	$\forall b,c,d,e,f_{\blacksquare}$	$(c \cup (((f \cap c) \cap (c \setminus f)) \cap d)) = (f \cup (e \cup (b \setminus (d \setminus c))))$
76	$\forall a,b,c,d,e,f_{\blacksquare}$	$(a \cap (((e \cup b) \setminus (a \cup d)) \cap ((f \cap a) \cap c))) \subseteq (((e \setminus (a \setminus e)) \cap ((d \cap a) \cup (c \cap f))) \cap (((c \setminus d) \cap (d \cap e)) \setminus b))$
77	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((c \cap (a \cup d)) \cap ((f \cup a) \setminus (a \setminus d))) \cup (f \cap ((b \setminus d) \cup (a \cup c)))) = ((a \setminus ((c \cup f) \cap (a \cup d))) \cup (e \cap ((d \setminus e) \setminus (f \cup c))))$
78	$\forall a,b,c,d,e,f_{\blacksquare}$	$((a \setminus ((b \cup f) \setminus (c \cap f))) \cup a) = ((e \cap ((f \setminus e) \cup (d \cup b))) \cup e)$
79	$\forall a,b,c,d,e,f_{\blacksquare}$	$d \subseteq ((((f \setminus a) \setminus (e \cap a)) \setminus d) \cap ((f \setminus (c \setminus b)) \cap ((d \cap b) \cup (f \cap e))))$
80	$\forall a,b,c,d,e,f_{\blacksquare}$	$c \subseteq (((a \cap e) \setminus (c \cap d)) \cap (((f \setminus d) \cup (d \cap c)) \cap ((c \cap d) \setminus (a \setminus b))))$
81	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((e \setminus (a \cup e)) \setminus b) \cup (c \setminus d)) \subseteq ((((a \cap d) \cup (c \cup a)) \cup ((f \setminus a) \setminus (c \setminus d))) \cup (((b \cup f) \setminus (d \setminus c)) \setminus ((c \setminus b) \setminus (d \cap b))))$
82	$\forall a,b,d,e_{\blacksquare}$	$(e \setminus (((b \cap a) \cap (\emptyset \cup e)) \cup d)) \neq ((((e \cap d) \cap (a \setminus \emptyset)) \setminus \emptyset) \cap a)$
83	$\forall a,b,c,d,e,f_{\blacksquare}$	$((e \setminus (a \cap (f \cup \emptyset))) \cap (((\emptyset \setminus (a \cup d)) \setminus d)) \neq (a \setminus (((f \setminus c) \cup (\emptyset \setminus b)) \cup c))$
84	$\forall a,b,c,d,f_{\blacksquare}$	$(d \cap (((c \cup f) \cup (b \cup a)) \setminus ((c \cup d) \setminus f))) = c$
85	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((b \cap c) \cap d) \cap ((b \cup e) \cap f)) \cup (((b \setminus f) \cap (c \cap a)) \cup (f \cup (d \cap b)))) \subseteq ((((d \setminus e) \setminus (a \setminus d)) \cup ((f \cup a) \cap (c \setminus e))) \setminus (d \setminus f))$
86	$\forall a,b,c,d,e_{\blacksquare}$	$((((e \cup b) \setminus (a \setminus b)) \cap ((a \setminus b) \setminus (c \cup b))) \cap (e \setminus ((b \cap c) \setminus (a \cap d)))) = d$
87	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((e \setminus c) \setminus (b \cap c)) \cup ((b \cup f) \cap (a \setminus d))) \cap (((d \setminus e) \setminus (e \setminus b)) \cap ((d \setminus a) \setminus (e \setminus f)))) = ((e \cap d) \cup (((b \cap c) \setminus (f \setminus e)) \setminus d))$
88	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((c \cap a) \cap (b \cup a)) \setminus (b \setminus (e \setminus f))) \cap (((c \setminus a) \setminus (b \cup e)) \setminus ((c \cup b) \cup (d \setminus f)))) \subseteq a$
89	$\forall a,b,c,d,e,f_{\blacksquare}$	$((c \cap ((b \cup e) \setminus e)) \cap a) \subseteq (((b \cap e) \setminus ((f \setminus c) \setminus (c \cup b))) \cap ((d \cup (a \setminus b)) \cap d))$
90	$\forall a,b,c,d,e,f_{\blacksquare}$	$(b \setminus (((f \cap c) \cap (c \setminus d)) \cap ((f \setminus b) \cap (c \setminus e)))) \subseteq ((((a \cap d) \setminus (d \setminus e)) \cup a) \cup (((c \cap a) \cap (d \setminus f)) \cap a))$
91	$\forall a,b,c,d,e,f_{\blacksquare}$	$((a \setminus ((b \cup c) \cup (f \cap c))) \setminus (((f \cup e) \cap (f \cup b)) \setminus ((d \setminus b) \cap e))) \subseteq ((f \cup e) \setminus (c \cup ((d \setminus b) \cup a)))$
92	$\forall a,b,c,d,e_{\blacksquare}$	$((c \cap ((a \cap b) \cup (c \cap e))) \cup ((d \setminus e) \setminus d)) \subseteq e$
93	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((a \setminus b) \cup (b \setminus c)) \cup (((c \setminus d) \cup (d \setminus e)) \cup ((f \setminus a) \cup (e \setminus f)))) \neq ((((a \setminus b) \cup c) \setminus a) \setminus c)$
94	$\forall a,b,c,d,e,f_{\blacksquare}$	$((c \setminus ((f \setminus b) \cup (f \cap a))) \cap ((f \cap (f \setminus b)) \cup (e \cup (d \cup a)))) \subseteq e$
95	$\forall a,b,c,d,e,f_{\blacksquare}$	$((e \cup ((a \cup f) \setminus c)) \cap (a \cap ((e \cap c) \cap (e \cap b)))) \subseteq ((e \cup ((c \cup d) \setminus (f \setminus d))) \cup ((a \cap (c \cup e)) \cap ((a \setminus b) \cup (c \setminus a))))$
96	$\forall a,b,d,e,f_{\blacksquare}$	$f \subseteq ((((a \cup b) \cap (e \cup f)) \cup d) \cup b)$
97	$\forall a,b,c,d,e,f_{\blacksquare}$	$(((((b \cap c) \cup (c \cup e)) \cup ((d \cap f) \setminus c)) \cup (((d \setminus e) \setminus (d \cap b)) \setminus ((d \cap e) \cup (f \cap b)))) \subseteq ((c \cap ((e \cup f) \setminus (b \cup a))) \cap (((c \setminus e) \setminus e) \setminus ((a \cap e) \cup (f \cup b))))$
98	$\forall a,d,e_{\blacksquare}$	$(((a \cap d) \cup ((e \cup d) \setminus a)) \cup a) \neq \emptyset$
99	$\forall a,b,c,d,e,f_{\blacksquare}$	$((((a \cup d) \cap (b \cap d)) \setminus (c \setminus (a \cap e))) \cap ((b \cup (e \cup a)) \setminus ((a \setminus c) \cup b))) \subseteq ((((f \cup d) \cup (a \setminus b)) \cup b) \cap b)$
100	$\forall a,b,c,d,ef_{\blacksquare}$	$((b \setminus ((d \cap a) \cup (c \cap d))) \setminus (((b \setminus a) \setminus (c \cap f)) \setminus ((c \setminus e) \cap (f \cap a)))) \subseteq ((b \cap ((e \cap a) \setminus (a \cup f))) \cap (b \cap ((f \cup c) \cup (b \setminus e))))$

Table A.3
Results for set equalities under no assumptions, i.e., unconditional

Prb	Vampire		Paradox		Oants			Prb	Vampire		Paradox		Oants			Prb	Vampire		Paradox		Oants		
	Time	Clauses	Time	Model	Result	Time	Steps		Time	Clauses	Time	Model	Result	Time	Steps		Time	Clauses	Time	Model	Result	Time	Steps
1	< .1	52	t/o	—	Proof	8	1	34	t/o	—	< .1	2	Ctrex.	22	2	67	t/o	—	< .1	2	Ctrex.	9	1
2	t/o	—	< .1	2	Ctrex.	26	2	35	t/o	—	< .1	2	Ctrex.	13	1	68	t/o	—	< .1	1	Ctrex.	8	1
3	t/o	—	< .1	2	Ctrex.	28	2	36	t/o	—	< .1	2	Ctrex.	8	1	69	t/o	—	< .1	2	Ctrex.	9	1
4	t/o	—	< .1	2	Ctrex.	8	1	37	t/o	—	< .1	1	Ctrex.	8	1	70	t/o	—	< .1	2	Ctrex.	9	1
5	< .1	47	t/o	—	Proof	9	1	38	t/o	—	< .1	2	Ctrex.	33	2	71	t/o	—	< .1	1	Ctrex.	8	1
6	t/o	—	< .1	2	Ctrex.	8	1	39	t/o	—	< .1	2	Ctrex.	8	1	72	t/o	—	< .1	2	Ctrex.	20	2
7	t/o	—	< .1	2	Ctrex.	8	2	40	t/o	—	< .1	2	Ctrex.	8	1	73	t/o	—	< .1	2	Ctrex.	71	2
8	t/o	—	< .1	2	Ctrex.	8	1	41	t/o	—	< .1	2	Ctrex.	29	2	74	t/o	—	< .1	2	Ctrex.	14	2
9	< .1	44	t/o	—	Proof	30	2	42	t/o	—	< .1	1	Ctrex.	8	1	75	t/o	—	< .1	2	Ctrex.	9	1
10	t/o	—	< .1	2	Ctrex.	14	2	43	t/o	—	< .1	2	Ctrex.	11	1	76	< .1	45	t/o	—	Proof	8	1
11	t/o	—	< .1	2	Ctrex.	10	1	44	t/o	—	< .1	2	Ctrex.	12	1	77	t/o	—	< .1	2	Ctrex.	9	1
12	t/o	—	< .1	2	Ctrex.	9	1	45	t/o	—	< .1	2	Ctrex.	11	1	78	t/o	—	< .1	2	Ctrex.	11	1
13	t/o	—	< .1	2	Ctrex.	10	1	46	t/o	—	< .1	2	Ctrex.	39	1	79	t/o	—	< .1	2	Ctrex.	9	1
14	t/o	—	t/o	—	Proof	17	2	47	t/o	—	< .1	2	Ctrex.	9	1	80	t/o	—	< .1	2	Ctrex.	8	1
15	t/o	—	< .1	2	Ctrex.	8	1	48	t/o	—	< .1	2	Ctrex.	8	1	81	< .1	53	t/o	—	Proof	8	1
16	t/o	—	< .1	2	Ctrex.	106	2	49	t/o	—	< .1	2	Ctrex.	9	1	82	t/o	—	< .1	1	Ctrex.	8	1
17	t/o	—	< .1	2	Ctrex.	9	1	50	t/o	—	< .1	2	Ctrex.	8	1	83	t/o	—	< .1	1	Ctrex.	8	1
18	< .1	54	t/o	—	Proof	11	1	51	t/o	—	< .1	2	Ctrex.	9	1	84	t/o	—	< .1	2	Ctrex.	9	1
19	t/o	—	< .1	2	Ctrex.	26	2	52	t/o	—	< .1	2	Ctrex.	9	1	85	t/o	—	< .1	2	Ctrex.	9	1
20	t/o	—	< .1	2	Ctrex.	19	2	53	t/o	—	< .1	2	Ctrex.	8	1	86	t/o	—	< .1	2	Ctrex.	9	1
21	t/o	—	< .1	2	Ctrex.	8	1	54	t/o	—	< .1	2	Ctrex.	8	1	87	t/o	—	< .1	2	Ctrex.	9	1
22	t/o	—	< .1	1	Ctrex.	8	1	55	t/o	—	< .1	1	Ctrex.	8	1	88	< .1	39	t/o	—	Proof	10	1
23	t/o	—	< .1	1	Ctrex.	8	1	56	< .1	44	t/o	—	Proof	10	1	89	t/o	—	< .1	2	Ctrex.	9	1
24	t/o	—	< .1	2	Ctrex.	10	1	57	t/o	—	< .1	2	Ctrex.	9	1	90	t/o	—	< .1	2	Ctrex.	9	1
25	t/o	—	< .1	2	Ctrex.	8	1	58	t/o	—	< .1	1	Ctrex.	8	1	91	t/o	—	< .1	2	Ctrex.	8	1
26	t/o	—	< .1	2	Ctrex.	22	2	59	t/o	—	< .1	2	Ctrex.	8	1	92	t/o	—	< .1	2	Ctrex.	9	1
27	t/o	—	< .1	2	Ctrex.	8	1	60	t/o	—	< .1	2	Ctrex.	9	1	93	t/o	—	< .1	1	Ctrex.	8	1
28	t/o	—	< .1	2	Ctrex.	9	1	61	t/o	—	< .1	2	Ctrex.	9	1	94	t/o	—	< .1	2	Ctrex.	9	1
29	< .1	46	t/o	—	Proof	8	1	62	t/o	—	< .1	2	Ctrex.	10	1	95	< .1	45	t/o	—	Proof	10	1
30	t/o	—	< .1	2	Ctrex.	34	2	63	< .1	39	t/o	—	Proof	11	1	96	t/o	—	< .1	2	Ctrex.	9	1
31	t/o	—	< .1	1	Ctrex.	8	1	64	t/o	—	t/o	—	Proof	67	2	97	t/o	—	< .1	2	Ctrex.	8	1
32	t/o	—	< .1	1	Ctrex.	8	1	65	t/o	—	< .1	1	Ctrex.	8	1	98	t/o	—	< .1	1	Ctrex.	8	1
33	t/o	—	< .1	2	Ctrex.	9	1	66	t/o	—	< .1	2	Ctrex.	10	1	99	< .1	45	t/o	—	Proof	8	1
																100	t/o	—	< .1	2	Ctrex.	9	1

Table A.4
Results for set equalities under uniqueness assumption

Prb	Vampire		Paradox		Oants			Prb	Vampire		Paradox		Oants			Prb	Vampire		Paradox		Oants		
	Time	Clauses	Time	Model	Result	Time	Steps		Time	Clauses	Time	Model	Result	Time	Steps		Time	Clauses	Time	Model	Result	Time	Steps
1	t/o	—	t/o	—	Proof	15	2	34	t/o	—	t/o	—	Ctrex.	30	3	67	t/o	—	t/o	—	Ctrex.	30	3
2	t/o	—	t/o	—	Ctrex.	35	3	35	t/o	—	t/o	—	Ctrex.	22	3	68	t/o	—	t/o	—	Ctrex.	48	3
3	t/o	—	t/o	—	Ctrex.	29	3	36	t/o	—	t/o	—	Ctrex.	31	3	69	t/o	—	t/o	—	Ctrex.	72	3
4	t/o	—	t/o	—	Ctrex.	27	3	37	t/o	—	t/o	—	Proof	40	2	70	t/o	—	t/o	—	Ctrex.	27	3
5	t/o	—	t/o	—	Proof	15	2	38	t/o	—	t/o	—	Ctrex.	32	3	71	t/o	—	t/o	—	Ctrex.	61	3
6	t/o	—	t/o	—	Ctrex.	34	3	39	t/o	—	t/o	—	Ctrex.	31	3	72	t/o	—	t/o	—	Ctrex.	27	3
7	t/o	—	.3	6	Ctrex.	8	1	40	t/o	—	t/o	—	Ctrex.	31	3	73	t/o	—	t/o	—	Ctrex.	28	3
8	t/o	—	t/o	—	Ctrex.	26	3	41	t/o	—	t/o	—	Ctrex.	31	3	74	t/o	—	.1	4	Ctrex.	8	1
9	t/o	—	t/o	—	Proof	14	2	42	t/o	—	t/o	—	Proof	17	2	75	t/o	—	t/o	—	Ctrex.	24	3
10	t/o	—	t/o	—	Ctrex.	33	3	43	t/o	—	t/o	—	Ctrex.	36	3	76	t/o	—	t/o	—	Proof	17	2
11	t/o	—	t/o	—	Ctrex.	23	3	44	t/o	—	t/o	—	Ctrex.	33	3	77	t/o	—	t/o	—	Ctrex.	33	3
12	t/o	—	t/o	—	Ctrex.	30	3	45	t/o	—	t/o	—	Ctrex.	35	3	78	t/o	—	t/o	—	Ctrex.	39	3
13	t/o	—	1.3	5	Ctrex.	8	1	46	t/o	—	t/o	—	Ctrex.	23	3	79	t/o	—	t/o	—	Ctrex.	29	3
14	t/o	—	t/o	—	Proof	16	2	47	t/o	—	t/o	—	Ctrex.	28	3	80	t/o	—	t/o	—	Ctrex.	27	3
15	t/o	—	t/o	—	Ctrex.	25	3	48	t/o	—	t/o	—	Ctrex.	32	3	81	t/o	—	t/o	—	Proof	16	2
16	t/o	—	t/o	—	Ctrex.	25	3	49	t/o	—	t/o	—	Ctrex.	26	3	82	t/o	—	t/o	—	Ctrex.	28	3
17	t/o	—	t/o	—	Ctrex.	23	3	50	t/o	—	t/o	—	Ctrex.	82	3	83	t/o	—	t/o	—	Ctrex.	37	3
18	t/o	—	t/o	—	Proof	16	2	51	t/o	—	t/o	—	Ctrex.	40	3	84	t/o	—	t/o	—	Ctrex.	24	3
19	t/o	—	.1	4	Ctrex.	8	1	52	t/o	—	t/o	—	Ctrex.	51	3	85	t/o	—	t/o	—	Ctrex.	30	3
20	t/o	—	t/o	—	Ctrex.	30	3	53	t/o	—	t/o	—	Ctrex.	108	3	86	t/o	—	t/o	—	Ctrex.	27	3
21	t/o	—	.3	5	Ctrex.	8	1	54	t/o	—	t/o	—	Ctrex.	49	3	87	t/o	—	t/o	—	Ctrex.	45	3
22	t/o	—	t/o	—	Ctrex.	55	3	55	t/o	—	t/o	—	Ctrex.	29	3	88	t/o	—	t/o	—	Proof	15	2
23	0.3	3529	t/o	—	Proof	9	1	56	t/o	—	t/o	—	Proof	44	2	89	t/o	—	t/o	—	Ctrex.	27	3
24	t/o	—	t/o	—	Ctrex.	39	3	57	t/o	—	t/o	—	Ctrex.	55	3	90	t/o	—	t/o	—	Ctrex.	29	3
25	t/o	—	t/o	—	Ctrex.	28	3	58	t/o	—	t/o	—	Ctrex.	55	3	91	t/o	—	t/o	—	Ctrex.	31	3
26	t/o	—	t/o	—	Ctrex.	33	3	59	t/o	—	t/o	—	Ctrex.	56	3	92	t/o	—	t/o	—	Ctrex.	23	3
27	t/o	—	.3	4	Ctrex.	11	1	60	t/o	—	$< .1$	4	Ctrex.	9	1	93	t/o	—	t/o	—	Proof	66	2
28	t/o	—	t/o	—	Ctrex.	53	3	61	t/o	—	$< .1$	4	Ctrex.	9	1	94	t/o	—	t/o	—	Ctrex.	39	3
29	t/o	—	t/o	—	Proof	15	2	62	t/o	—	$< .1$	4	Ctrex.	9	1	95	t/o	—	t/o	—	Proof	15	2
30	t/o	—	t/o	—	Ctrex.	30	3	63	t/o	—	t/o	—	Proof	91	2	96	t/o	—	.2	5	Ctrex.	8	1
31	t/o	—	.2	6	Ctrex.	9	1	64	t/o	—	t/o	—	Proof	41	2	97	t/o	—	t/o	—	Ctrex.	39	3
32	33.5	54 920	t/o	—	Proof	48	2	65	0.5	5421	t/o	—	Proof	10	1	98	t/o	—	$< .1$	4	Ctrex.	8	1
33	t/o	—	t/o	—	Ctrex.	25	3	66	t/o	—	.1	4	Ctrex.	8	1	99	t/o	—	t/o	—	Proof	14	2
																100	t/o	—	t/o	—	Ctrex.	29	3

Table A.5
Results for set equalities under disjoint assumption

Prb	Vampire		Paradox		Oants			Prb	Vampire		Paradox		Oants			Prb	Vampire		Paradox		Oants		
	Time	Clauses	Time	Model	Result	Time	Steps		Time	Clauses	Time	Model	Result	Time	Steps		Time	Clauses	Time	Model	Result	Time	Steps
1	t/o	—	t/o	—	Proof	55	3	34	t/o	—	< .1	2	Ctrex.	10	1	67	t/o	—	< .1	2	Ctrex.	9	1
2	t/o	—	< .1	2	Ctrex.	10	1	35	t/o	—	< .1	2	Ctrex.	10	1	68	t/o	—	< .1	1	Ctrex.	8	1
3	t/o	—	< .1	2	Ctrex.	10	1	36	t/o	—	< .1	2	Ctrex.	11	1	69	t/o	—	< .1	2	Ctrex.	10	1
4	t/o	—	< .1	2	Ctrex.	9	1	37	t/o	—	< .1	1	Ctrex.	8	1	70	t/o	—	< .1	2	Ctrex.	9	1
5	65.4	495 784	t/o	—	Proof	14	2	38	t/o	—	< .1	2	Ctrex.	10	1	71	t/o	—	< .1	1	Ctrex.	8	1
6	t/o	—	< .1	2	Ctrex.	10	1	39	t/o	—	t/o	—	Proof	48	2	72	t/o	—	< .1	2	Ctrex.	9	1
7	t/o	—	t/o	—	Proof	43	2	40	t/o	—	< .1	2	Ctrex.	10	1	73	t/o	—	< .1	2	Ctrex.	9	1
8	t/o	—	< .1	2	Ctrex.	10	1	41	t/o	—	< .1	2	Ctrex.	10	1	74	t/o	—	< .1	2	Ctrex.	9	1
9	t/o	—	t/o	—	Proof	92	2	42	t/o	—	< .1	1	Ctrex.	9	1	75	t/o	—	< .1	2	Ctrex.	9	1
10	t/o	—	< .1	2	Ctrex.	10	1	43	t/o	—	< .1	2	Ctrex.	13	2	76	t/o	—	t/o	—	Proof	25	2
11	t/o	—	< .1	2	Ctrex.	10	1	44	t/o	—	< .1	2	Ctrex.	8	1	77	t/o	—	< .1	2	Ctrex.	9	1
12	t/o	—	t/o	—	Proof	52	2	45	t/o	—	.1	2	Ctrex.	10	1	78	t/o	—	< .1	2	Ctrex.	9	1
13	6.3	62 426	t/o	—	Proof	17	2	46	t/o	—	t/o	—	Proof	15	2	79	t/o	—	< .1	2	Ctrex.	9	1
14	t/o	—	t/o	—	Proof	48	2	47	t/o	—	< .1	2	Ctrex.	11	1	80	t/o	—	< .1	2	Ctrex.	9	1
15	t/o	—	< .1	2	Ctrex.	8	1	48	t/o	—	< .1	2	Ctrex.	10	1	81	t/o	—	t/o	—	Proof	65	2
16	t/o	—	t/o	—	Proof	16	2	49	t/o	—	< .1	2	Ctrex.	10	1	82	t/o	—	< .1	1	Ctrex.	8	1
17	t/o	—	< .1	2	Ctrex.	10	1	50	t/o	—	< .1	2	Ctrex.	10	1	83	t/o	—	< .1	1	Ctrex.	8	1
18	t/o	—	t/o	—	Proof	17	2	51	t/o	—	< .1	2	Ctrex.	9	1	84	t/o	—	< .1	2	Ctrex.	56	2
19	t/o	—	< .1	2	Ctrex.	10	1	52	t/o	—	.1	2	Ctrex.	9	1	85	t/o	—	< .1	2	Ctrex.	25	1
20	t/o	—	< .1	2	Ctrex.	11	1	53	t/o	—	t/o	—	Proof	49	2	86	t/o	—	< .1	2	Ctrex.	9	1
21	31.3	168 393	t/o	—	Proof	14	2	54	t/o	—	t/o	—	Proof	91	2	87	t/o	—	t/o	—	Proof	18	2
22	t/o	—	< .1	1	Ctrex.	8	1	55	t/o	—	< .1	1	Ctrex.	8	1	88	t/o	—	t/o	—	Proof	18	2
23	t/o	—	< .1	1	Ctrex.	8	1	56	t/o	—	t/o	—	Proof	41	2	89	17.6	170 631	t/o	—	Proof	15	2
24	t/o	—	.1	2	Ctrex.	10	1	57	t/o	—	< .1	2	Ctrex.	9	1	90	t/o	—	< .1	2	Ctrex.	8	1
25	t/o	—	t/o	—	Proof	53	2	58	t/o	—	< .1	1	Ctrex.	8	1	91	t/o	—	< .1	2	Ctrex.	8	1
26	t/o	—	< .1	2	Ctrex.	10	1	59	t/o	—	< .1	2	Ctrex.	9	1	92	t/o	—	t/o	—	Proof	14	2
27	t/o	—	< .1	2	Ctrex.	12	1	60	t/o	—	< .1	2	Ctrex.	9	1	93	t/o	—	< .1	1	Ctrex.	8	1
28	t/o	—	.1	2	Ctrex.	10	1	61	t/o	—	< .1	2	Ctrex.	9	1	94	t/o	—	t/o	—	Proof	45	2
29	t/o	—	t/o	—	Proof	16	2	62	t/o	—	< .1	2	Ctrex.	9	1	95	t/o	—	t/o	—	Proof	48	2
30	34.7	276 481	t/o	—	Proof	21	2	63	t/o	—	t/o	—	Proof	17	2	96	t/o	—	< .1	2	Ctrex.	48	2
31	t/o	—	< .1	1	Ctrex.	8	1	64	t/o	—	—	—	Proof	20	2	97	t/o	—	.1	2	Ctrex.	38	1
32	t/o	—	< .1	1	Ctrex.	8	1	65	0.1	4040	t/o	—	Proof	9	1	98	t/o	—	< .1	1	Ctrex.	8	1
33	t/o	—	< .1	2	Ctrex.	10	1	66	t/o	—	< .1	2	Ctrex.	8	1	99	t/o	—	t/o	—	Proof	17	2
																100	t/o	—	t/o	—	Proof	60	2

(Ctrex.), depending on the validity of the statement, the time OANTS needed, and the number of reasoning steps OANTS had to perform before either VAMPIRE or PARADOX could discharge the problem. Observe that in some cases, OANTS needs two steps, despite PARADOX or VAMPIRE being able to solve the initial problem in insignificant time. This is due to possible communication delays via the networked file system, and a second run might yield a slightly different result.

References

[1] P. Andrews, General models, descriptions and choice in type theory, Journal of Symbolic Logic 37 (2) (1972) 385–394.
[2] P. Andrews, An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Applied Logic Series, vol. 27, Kluwer, 2002.
[3] P.B. Andrews, Resolution in type theory, Journal of Symbolic Logic 36 (3) (1971) 414–432.
[4] C. Benzmüller, Equality and extensionality in higher-order theorem proving, PhD thesis, Univ. des Saarlandes, Germany, 1999.
[5] C. Benzmüller, Comparing approaches to resolution based higher-order theorem proving, Synthese 133 (1–2) (2002) 203–235.
[6] C. Benzmüller, M. Jamnik, M. Kerber, V. Sorge, Towards concurrent resource managed deduction, Tech. Rep. CSRP-99-17, University of Birmingham, School of Computer Science, 1999; URL: ftp.cs.bham.ac.uk/pub/tech-reports/1999/CSRP-99-17.ps.gz.
[7] C. Benzmüller, M. Jamnik, M. Kerber, V. Sorge, Experiments with an Agent-Oriented Reasoning System, in: Proc. of KI 2001, in: LNAI, vol. 2174, Springer, 2001, pp. 409–424.
[8] C. Benzmüller, M. Kohlhase, LEO—a higher-order theorem prover, in: Proc. of CADE-15, in: LNAI, vol. 1421, Springer, 1998, pp. 139–143.
[9] C. Benzmüller, V. Sorge, A blackboard architecture for guiding interactive proofs, in: Proc. of AIMSA'98, in: LNAI, vol. 1480, Springer, 1998, pp. 102–114.
[10] C. Benzmüller, V. Sorge, OANTS—An open approach at combining interactive and automated theorem proving, in: Proc. of Calculemus-2000, AK Peters, 2001, pp. 81–97.
[11] C. Benzmüller, V. Sorge, M. Jamnik, M. Kerber, Can a higher-order and a first-order theorem prover cooperate?, in: F. Baader, A. Voronkov (Eds.), LPAR, in: LNCS, vol. 3452, Springer, 2005, pp. 415–431.
[12] M. Bishop, P. Andrews, Selectively instantiating definitions, in: Proc. of CADE-15, in: LNAI, vol. 1421, Springer, 1998, pp. 365–380.
[13] M. Bonacina, A taxonomy of parallel strategies for deduction, Annals of Mathematics and Artificial Intelligence 29 (1–4) (2000) 223–257.
[14] A. Bond, L. Gasser, An analysis of problems and research in DAI, in: A. Bond, L. Gasser (Eds.), Readings in Distributed Artificial Intelligence, Morgan Kaufmann, 1988, pp. 3–35.
[15] C.E. Brown, Set comprehension in Church's type theory, PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, USA, 2004.
[16] L. Cheikhrouhou, V. Sorge, $\mathcal{PDS}$—A three-dimensional data structure for proof plans, in: Proc. of ACIDCA'2000, 2000.
[17] H. de Nivelle, The Bliksem theorem prover, Version 1.12. Max-Planck-Institut, 1999, http://www.mpi-sb.mpg.de/bliksem/manual.ps.
[18] J. Denzinger, D. Fuchs, Cooperation of heterogeneous provers, in: Proc. of IJCAI-16, Morgan Kaufmann, 1999, pp. 10–15.
[19] J. Denzinger, M. Fuchs, Goal oriented equational theorem proving using team work, in: Proc. of KI-94, in: LNAI, vol. 861, Springer, 1994, pp. 343–354.
[20] M. Fisher, A. Ireland, Multi-agent proof-planning, in: CADE-15 Workshop "Using AI Methods in Deduction", 1998.
[21] H. Ganzinger, J. Stuber, Superposition with equivalence reasoning and delayed clause normal form transformation, in: Proc. of CADE-19, in: LNAI, vol. 2741, Springer, 2003, pp. 335–349.
[22] J. Hurd, An LCF-style interface between HOL and first-order logic, in: Automated Deduction—CADE-18, in: LNAI, vol. 2392, Springer, 2002, pp. 134–138.
[23] M. Kerber, On the representation of mathematical concepts and their translation into first order logic, PhD thesis, Universität Kaiserslautern, Germany, 1992.
[24] M. Kerber, M. Kohlhase, V. Sorge, Integrating computer algebra into proof planning, Journal of Automated Reasoning 21 (3) (1998) 327–355.
[25] A. Meier, TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level, in: Proc. of CADE-17, in: LNAI, vol. 1831, Springer, 2000, pp. 460–464.
[26] J. Meng, L. Paulson, Experiments on supporting interactive proof using resolution, in: Proc. of IJCAR-04, in: LNCS, vol. 3097, Springer, 2004, pp. 372–384.
[27] J. Meng, C. Quigley, L.C. Paulson, Automation for interactive proof: First prototype, Inf. Comput. 204 (10) (2006) 1575–1596.
[28] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, K. Mayr, SETHEO and e-SETHEO—the CADE-13 systems, Journal of Automated Reasoning 18 (2) (1997) 237–246.
[29] J. Müller, A cooperation model for autonomous agents, in: Proc. of the ECAI'96 Workshop Intelligent Agents, in: LNAI, vol. 1193, Springer, 1997, pp. 245–260.
[30] H. Nii, E. Feigenbaum, J. Anton, A. Rockmore, Signal-to-symbol transformation: HASP/SIAP case study, AI Magazine 3 (2) (1982) 23–35.
[31] D. Pastre, Muscadet2.3: A knowledge-based theorem prover based on natural deduction, in: Proc. of IJCAR-01, in: LNAI, vol. 2083, Springer, 2001, pp. 685–689.
[32] A. Riazanov, A. Voronkov, The design and implementation of vampire, AI Communications 15 (2–3) (2002) 91–110.
[33] J. Rice, The ELINT application on polygon: The architecture and performance of a concurrent blackboard system, in: Proc. of IJCAI-11, Morgan Kaufmann, 1989, pp. 212–220.
[34] R.C. Sekar, I.V. Ramakrishnan, A. Voronkov, Term indexing, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, MIT Press, 2001.
[35] V. Sorge, Non-trivial symbolic computations in proof planning, in: Proc. of FROCOS 2000, in: LNCS, vol. 1794, Springer, 2000, pp. 121–135.

[36] V. Sorge, Oants—a blackboard architecture for the integration of reasoning techniques into proof planning, PhD thesis, Univ. des Saarlandes, Germany, 2001.
[37] G. Sutcliffe, C. Suttner, The TPTP problem library: CNF release v1.2.1, Journal of Automated Reasoning 21 (2) (1998) 177–203.
[38] G. Sutcliffe, J. Zimmer, S. Schulz, Communication formalisms for automated theorem proving tools, in: Proc. of IJCAI-18 Workshop on Agents and Automated Reasoning, 2003.
[39] G. Weiss, Prologue, in: G. Weiss (Ed.), Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, 1999, pp. 1–23.
[40] M. Wooldridge, Intelligent agents, in: G. Weiss (Ed.), Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, 1999, pp. 27–77.