Experiments On Supporting Interactive Proof Using Resolution

Jia Meng and Lawrence C. Paulson

Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge CB3 0FD (UK)
{jm318,lp15}@cam.ac.uk

Abstract. Interactive theorem provers can model complex systems, but require much effort to prove theorems. Resolution theorem provers are automatic and powerful, but they are designed to be used for very different applications. This paper reports a series of experiments designed to determine whether resolution can support interactive proof as it is currently done. In particular, we present a sound and practical encoding in first-order logic of Isabelle's type classes.

1 Introduction

Interactive proof tools such as HOL [4], Isabelle [8] and PVS [10] have been highly successful. They have been used for verifying hardware, software, protocols, and so forth. Unfortunately, interactive proof requires much effort from a skilled user. Many other tools are completely automatic, but they cannot be used to verify large systems. Can we use automatic tools to improve the automation of interactive provers?

In this paper, we report a year's experiments aimed at assessing whether resolution theorem provers can assist interactive provers. For such an integration to be effective, we must bridge the many differences between a typical interactive theorem prover and a resolution theorem prover:

- higher-order logic versus first-order logic
- polymorphically typed (or other complicated type system) versus untyped
- natural deduction or sequent formalism versus clause form

Particularly difficult is the problem of coping with large numbers of previously proved lemmas. In interactive proof, the user typically proceeds by proving hundreds or thousands of lemmas that support later parts of the verification. Ideally, the user should not have to select the relevant lemmas manually, but too many irrelevant facts may overwhelm the automatic prover.

A number of other researchers have attempted to combine interactive and automatic theorem provers. The HOL system has for many years included a model elimination theorem prover, which recently Hurd has attempted to improve upon [6]. The Coq system has also been interfaced with an automatic first-order prover [2]. These approaches expect the user to pick relevant lemmas

manually. Another attempt, using the KIV system [1], includes an automatic mechanism for discarding irrelevant lemmas. This approach has attractions, and the overall performance might be improved by using a more powerful automatic prover.

Closest to our conception is the Ωmega system [18]. One key idea we share is that assistants should run as background processes. The interactive user should not have to notice that a certain tool may prove a certain subgoal: it should be attempted automatically. However, there are important differences. Ωmega is designed to support working mathematicians, and it has been combined with a large number of other reasoning tools. Our aim is to support formal verification, and we are trying to achieve the best possible integration with one or two other reasoning tools. Creative mathematics and verification are different applications: the mathematician's main concern is to arrive at the right definitions, while the verifier's main concern is to cope with fixed but enormous definitions.

Our work is by no means complete. Our experiments are designed to identify the main obstacles to an effective integration between an interactive and automatic prover. We have taken typical problems that can be solved in Isabelle, either using Isabelle's automatic tools or by short sequences of proof commands. We have converted these problems into clause form and supplied them to resolution provers (Vampire or SPASS). Crucially, we have given the resolution prover a large set of axioms, corresponding to previous lemmas that would by default be available to Isabelle's own automatic tools. One of our findings is that even the best resolution provers sometimes founder when given large sets of irrelevant axioms. We have been able to add several hard problems to the TPTP library [19].[1] We hope that the engineers of resolution provers will make progress on the problem of relevance, and we have already had excellent cooperation from the Vampire team. We have also developed a way of modelling Isabelle's type class system in first-order logic.

Paper outline. We present background information on Isabelle and Vampire (§2). We briefly describe earlier experiments involving an untyped formalism, namely set theory (§3). We then describe new experiments involving a polymorphically typed formalism, namely higher-order logic (§4). We finally offer some conclusions (§5).

2 Background: Isabelle and Vampire

Isabelle [8] is an interactive theorem prover. Unusually, Isabelle is generic: it supports a multiplicity of logics. The most important of these is higher-order logic, which is also the basis of the HOL system and PVS. Isabelle also supports Zermelo-Fraenkel set theory [14], which is an untyped formalism based upon first-order logic.

Isabelle provides substantial automation. Its reasoning tactics include the following:

[1] SET787-1.p, SET787-2.p, COL088-1.p to COL0100-2.p

– *simp* is the simplifier, which performs conditional rewriting augmented by other code, including a decision procedure for linear arithmetic.
– *blast* is a sort of generic tableaux theorem prover. It performs forward and backwards chaining using any lemmas supplied by the user [13].
– *auto* is a naive combination of the previous two tactics. It interleaves rewriting and chaining. However, this treatment of equality is primitive compared with that provided by a good resolution prover.

Many other Isabelle tactics are variants of those described above. Although these tactics are powerful, the user has to choose which one is appropriate and invoke it manually. A resolution prover can perform all the types of reasoning done by these tactics. Can a resolution prover, running in the background, replace all the calls to *simp*, *blast* and *auto*?

Isabelle's tactics have one major advantage: they let the user declare lemmas for them to use, and they easily cope with hundreds of such lemmas. After proving an equality, the user can declare it as a *simplification rule*. If a lemma looks suitable for forwards chaining, the user can declare it as such, and similarly for backwards chaining. The accumulation of many such declarations greatly improves the automation available to the user.

Certain other lemmas are not declared permanently, but supplied when needed for particular proofs. For example, distributive laws like $x*(y+z) = x*y + x*z$ should not usually be declared as simplification rules because they can cause an exponential blow up. Such lemmas can be named explicitly in an invocation of *simp*. Similarly, a transitivity law like $x < y \Longrightarrow y < z \Longrightarrow x < z$ should not usually be declared as suitable for backward chaining because it produces an explosion of subgoals. It can be mentioned in a call to *blast*.

If a resolution prover is to replace the role played by *simp*, *blast* and *auto*, then it must be able to do something appropriate with such lemma declarations. Equalities can be recognized by their syntactic form. The Vampire developers have kindly extended Vampire: version 6.03 allows annotations on literals to specify that a clause should be used for forward or backward chaining. This extension improves Vampire's performance on our examples. The main problem, which appears to affect most resolution provers, is that the sheer number of default lemmas makes the search space explode.

It is not enough to prove the theorems: we must also convince Isabelle that they have been proved. Isabelle, like HOL, minimises the risk of soundness bugs by allowing only a small kernel of the system to assert theorems. The translations between Isabelle and the resolution provers could easily contain errors, even if the provers themselves are sound. Therefore, we would like possible to translate the resolution proof back into a native one, as Hurd has already done between Gandalf and HOL [5]. This could be done by implementing Isabelle versions of the rules used in automatic proofs, such as resolution and paramodulation. Isabelle could then re-play the proof found by the automatic tool. The translated proof could also be stored, allowing future executions of the proof script to run without the automatic tool. Another use of the resolution proof is to identify the relevant Isabelle lemmas; that information might be valuable to users.

Vampire [16] and SPASS [20] are the provers we have used for our experiments. These are leading resolution provers that have done well in recent CADE ATP System Competitions. Our objective to support integration with any resolution prover that outputs explicit proofs. We convert Isabelle's formalisms into untyped first-order logic rather than expecting the resolution prover to support them.

3 Formalising Untyped Isabelle/ZF in FOL

As Meng has reported elsewhere [7], our first experiments concerned translating Isabelle/ZF formulas into first-order logic (FOL) in order to examine whether the use of resolution was practial. These experiments consisted of taking existing Isabelle proof steps performed by `simp`, `blast`, `auto`, etc., trying to reproduce them using Vampire. All simplification rules and backward/forward chaining rules of the current Isabelle context were translated to clause form. The goals were negated, converted to clauses and finally sent to Vampire.

3.1 Translation Issues

We translate those Isabelle/ZF formulas that are already in FOL form into conjunctive normal form (CNF) directly. However, there are some ZF formulas that lie outside first-order logic. We need to reformulate them into FOL form before CNF transformation. Some of these issues are particular to Isabelle alone. For example, set intersection satisfies the equality

$$(c \in A \cap B) = (c \in A \wedge c \in B)$$

An equation between Boolean terms is obviously not first-order. Moreover, Isabelle represents the left-to-right implication in a peculiar fashion related to its encoding of the sequent calculus [11]. Our translation has to recognize this encoding and translate it to the corresponding implication, which in this case is

$$\forall c\, A\, B\, [c \in A \cap B \rightarrow (c \in A \wedge c \in B)]$$

We also need to remove ZF terms, such as $\bigcup_{x \in A} B(x)$, since they are not present in first-order logic. Let $\phi(Z)$ be a formula containing a free occurrence of some term Z, and let $\phi(v)$ be the result of replacing Z by the variable v. Then $\phi(Z)$ is equivalent (in ZF) to

$$\exists v\, [\phi(v) \wedge \forall u\, [u \in v \leftrightarrow u \in Z]]$$

This transformation allows any occurrence of the term Z to be forced into a context of the form $u \in Z$. In this case, Z is $\bigcup_{x \in A} B(x)$, and translations can then replace $u \in \bigcup_{x \in A} B(x)$ by $\exists x\, [x \in A \wedge u \in B(x)]$.

Some other translation issues also arose during our experiments. The subset relation $A \subseteq B$ is equivalent to $\forall x\, (x \in A \rightarrow x \in B)$: this reduces the subset

relation to the membership relation. Experiments [7] showed that Vampire can find a proof much more quickly if the subset relation is replaced by its equivalent membership relation. This is probably because during most of the complex proofs, subset relations have to be reduced to equivalent membership relations anyway.

3.2 Experimental Results

An aim of these experiments was to find out whether the Isabelle/ZF-Vampire integration can prove goals that were proved by Isabelle's built-in tools such as *simp*, *blast* and *auto*. (Obviously, our ultimate goal is to supersede these tools, not merely to equal them.) We used three theory files:

- `equalities.thy`: proofs of many simple set equalities
- `Comb.thy`: a development of combinatory logic similar to the Isabelle/HOL version described by Paulson [11]
- `PropLog.thy`: a development of propositional logic

Thirty-seven lemmas (63 separate goals) were taken from Isabelle/ZF's theory files. The resolution prover could prove 52 goals out of 63 in the presence of a large set of axioms: 129 to 160 axiom clauses.

We also tried to examine if this integration can prove goals that cannot be proved by Isabelle's built-in tools and hence reduce user interaction. These goals were originally proved by a series of proof steps. Fifteen lemmas from `Comb.thy` and `PropLog.thy` were examined. Vampire proved ten lemmas.

Meng [7] gives a more detailed presentation of the experimental results.

3.3 Performance on Large Axiom Sets

An issue that arose from our ZF experiments is that many problems could only be proved for a minimal set of axioms and not with the full set of default axioms. Recall that one of our objectives is to preserve Isabelle's policy of not usually requiring the user to identify which previous lemmas should be used.

We took fifteen problems that seemed difficult in the presence of the full axiom set. We offered them to Geoff Sutcliffe for inclusion in the TPTP Library [19]. He kindly ran experiments using three provers (E, SPASS and Vampire) together with a tool he was developing for the very purpose of eliminating redundant axioms. Gernot Stenz ran the same problems on E-SETHEO, because that system is not available for downloading. Finally, Meng herself attempted the problems using both Vampire and SPASS. Thus, we made $15 \times 6 = 90$ trials altogether. These trials were not uniform, as they involved different hardware and different resource limits, but they are still illustrative of our difficulty.

Of the fifteen problems, only five could be proved. Two of them were proved in under two seconds by E-SETHEO. Sutcliffe proved the same two using SPASS, as well as a third problem; these took 43, 75 and 154 seconds, respectively. Meng was able to prove two other problems using a new version of Vampire (6.03, which

supports literal annotations; version 5.6 could not prove them). Thus, only seven of the ninety proof attempts succeeded.

Unfortunately, the hardest problems arose from proofs using the technique of *rule inversion*, which is important for reasoning about operational semantics. Rule inversion is a form of case analysis that involves identifying which of the many rules of an operational semantics definition may have caused a given event. Isabelle's `blast` method handles such proofs easily, but converting the case analysis rule to clause form yields an explosion: 135 clauses in one simple case. We have been able to reduce this number by various means, but the proofs remain difficult.

4 Formalising Typed Isabelle/HOL in FOL

Our previous experiments were based on Isabelle/ZF in order to avoid the complications of types. However, few Isabelle users use ZF. If our integration is to be useful, it must support higher-order logic, which in turn requires a sound modelling of the intricacies of Isabelle's type system.

4.1 Types and Sorts in Isabelle/HOL

Isabelle/HOL [8] supports *axiomatic type classes* [21]. A *type class* is a set of types for which certain operations are defined. An *axiomatic* type class has a set of axioms that must be satisfied by its instances: types belonging to that class. If a type τ belongs to a class C then it is written as $\tau :: C$.

A type class C can be a subclass of another type class D, if all axioms of D can be proved in C. If a type τ is an instance of C then it is an instance of D as well. Furthermore, a type class may have more than one direct superclass. If C is a subclass of both D_1 and D_2 then C is subset of intersection of D_1 and D_2. The intersection of type classes is called a *sort*. For example, Isabelle/HOL defines the type class of linear orders (`linorder`) to be a subclass of the type class of partial orders (`order`).

```
axclass linorder < order
  linorder_linear: "x≤ y ∨ y≤ x"
```

Now, to assert that type `real` is an instance of class `linorder`, we must show that the corresponding instance of the axiom `linorder_linear` holds for that type.

```
instance real :: linorder
  proof ... qed
```

Axiomatic type classes allows meaningful overloading of both operators and theorems about those operators. We can prove theorems for class `linorder`, and they will hold for all types in that class, including types defined in future Isabelle sessions. We can prove a theorem such as $(-a) \times (-b) = a \times b$ in type class `ring` and declare it as a simplification rule, where it will affect numeric types such as `int`, `rat`, `real` and `complex`. Type checking remains decidable, for it is the user's

responsibility to notice that a type belongs to a certain class and to declare it as such, providing the necessary proofs. Of course, full type checking must be performed, including checking of sorts, since a theorem about linear orderings cannot be assumed to hold for arbitrary orderings.

FOL automatic provers usually do not support types. However, when a HOL formula is translated to FOL clauses, this type and class information should be kept. Not only is the type information essential for soundness, but it can also reduce the search space significantly.

We can formalise axiomatic type classes in terms of FOL predicates, with types as FOL terms. Each type class corresponds to a unary predicate. If a type τ is an instance of a class C, then $C(\tau)$ will be true. Therefore, the subclass relation can be expressed by predicate implications. Taking the class `linorder` as an example, we have the formula

$$\forall \tau\, [linorder(\tau) \rightarrow order(\tau)]$$

Sort information for multiple inheritance can be handled similarly. $\tau :: C_1, \ldots, C_n$ can be expressed as $C_1(\tau) \wedge \ldots \wedge C_n(\tau)$.

A further complication of type classes concerns type constructors such as `list`. A list can be linearly ordered (by the usual lexicographic ordering) if we have a linear ordering of the list element types. This statement can be formalised in Isabelle as

```
instance list :: (linorder) linorder
  proof ... qed
```

We represent this in FOL by $\forall \tau\, [linorder(\tau) \rightarrow linorder(list(\tau))]$.

4.2 Polymorphic Operators Other Than Equality

Many predicates and functions in Isabelle/HOL are polymorphic. In our type class example, the relation $\leq$ is polymorphic. Its type is $\alpha \rightarrow \alpha \rightarrow bool$, where α is a type variable of class `ord`. The effect is to allow $\leq$ to be applied only if its arguments belong to type class `ord`. This polymophic type can be specialised when $\leq$ is applied to different arguments. When applied to sets, its type will be $\alpha\ set \rightarrow \alpha\ set \rightarrow bool$, whereas for natural numbers its type will be $nat \rightarrow nat \rightarrow bool$.

To formalise this type information of operators, for each operator (except constants and equality), we include its type as an additional argument. Constants do not need to carry type information: their types should be inferred automatically.

For example, axiom `linorder_linear` will be translated to

$$\forall \tau\ x\ y\, [linorder(\tau) \rightarrow (le(F(\tau, F(\tau, bool)), x, y) \vee le(F(\tau, F(\tau, bool)), y, x))]$$

where le is the predicate $\leq$ in prefix form and F is the function type constructor.

4.3 Types for Equalities

In Isabelle/HOL, equality is polymophic. When we prove the equality $A = B$, we may have to use different inference rules depending on the type of A and B. However, Vampire's built-in equality `equal` does not allow us to include the type of A and B as an extra argument.

There could be several ways to solve the problem. We could define a new equality predicate taking three arguments instead of two. However, automatic provers usually treat equality specially, giving much better performance than could be achieved with any user defined equality literal. Experiments showed that Vampire needed excessive time to find proofs involving this new equality predicate.

Therefore, we decided to use the built-in equality predicate while embedding the type information in its arguments. Instead of writing the formula $equal(A, B)$, we can include the type as

$$equal(typeinfo(A, \tau), typeinfo(B, \tau))$$

where τ is the type of A and B. The value of τ will determine which inference rule should be used to prove an equality. Equalities in previously proved lemmas and conjectures will be translated into *equal* in this format with all type information included. In addition, the axiom

$$equal(typeinfo(A, \tau), typeinfo(B, \tau)) \rightarrow equal(A, B)$$

is sometimes required for paramodulation. Its effect is to strip the types away. In experiments, this type embedding gave a reasonable performance.

Equalities between boolean-valued terms are simply viewed as two implications in the transformation to clause form.

4.4 Vampire Settings

In Isabelle, rules usually have information indicating whether they should be used for forward chaining (elimination rules) or backward chaining (introduction rules). We would like this information to be preserved after rules are translated into clauses. We attempted to accomplish this by assigning weights and precedences to functions and predicates to indicate which literals should be eliminated sooner; this information gives an ordering on literals, which Vampire computes using the Knuth-Bendix Ordering (KBO) [15]. However, since the resulting KBO is a partial ordering on terms with variables, it does not match our requirements exactly.

The Vampire developers gave us a new version of Vampire (v6.03), with syntax to specify which literal in a clause should be selected for resolution. This syntax is an extension of TPTP syntax. Any positive literal that should be resolved first will be tagged with `+++`, and similarly a negative literal should be tagged with `---`. Many lemmas could only be proved with this facility, which supports Isabelle's notions of forward and backward chaining.

Furthermore, Vampire provides many settings that can be specified by users to fine tune its performance according to different types of problems. We carried out many experiments in order to find the best combination of these settings. The experimental results [7] showed that six combinations of settings worked well under various circumstances. They were written to six setting files so that we can run six processes in parallel if necessary. We intend to experiment with different strategies for making the best use of the available processors. For example, if we have only one processor, then we may adopt a time-sharing mechanism, assigning suitable priorities and time limits to processes.

One particular issue is Set-of-Support (`SOS`). This strategy, which according to Chang and Lee [3] dates from 1965, forbids resolution steps that take all their clauses from a part of the clause set known to be satisfiable. This strategy seems ideal for problems such as ours that have a large axiom set, since it prevents deductions purely involving the axioms. However, it is incomplete in the presence of modern ordering heuristics and therefore is normally switched off. We have found that some lemmas can be proved only if `SOS` is on.

4.5 Experimental Results

We used the same general approach as we did in our earlier experiments on untyped ZF formulas. Isabelle/HOL lemmas were chosen, each of them usually presenting more than one goal to Vampire. The combination of the six setting files was used. The time limit for each proof attempt was 60sec. We used *formula renaming* [9] before the CNF transformation in order to minimize the number of clauses.

For typed Isabelle/HOL formulas, the inclusion of type information also helps to cut down the search space significantly. For instance if we want to prove subset relation between two sets: $X \leq Y$, its typed formula will be

$$le(F(set(\tau), F(set(\tau), bool)), X, Y)$$

Clearly inference rules such as

$$le(F(nat, F(nat, bool)), A, B)$$

will be ignored as the types of le will not match. Some experiments have been carried out to demonstrate the benefit of using such type information.

The primary aim of these experiments was to investigate whether this encoding of type information is practical for resolution proofs. This set of experiments took 56 lemmas (108 goals) from Isabelle/HOL theory files and tried to reproduce the proofs. We used the following theories:

- `Multiset.thy`: a development of multisets
- `Comb.thy`: combinatory logic formalized in higher-order logic
- `List_Prefix.thy`: a prefixing relation on lists
- `Message.thy`: a theory of messages for security protocol verification [12]

Theory	*Number of Lemmas*	*Number of Goals*	*Number of Goals Proved*
`Multiset`	3	3	3
`Comb`	18	29	24
`List_Prefix`	7	8	8
`Message`	28	68	62

Table 1. Number of Goals Proved for Typed Lemmas

Around 70 to 130 axiom clauses were used. Ninety-seven goals were proved using this typed formalism, as shown in Table 1.

Eleven lemmas from `Message.thy` cannot be proved by Isabelle's classical reasoners directly. Either they consist of more than one proof command or they explicitly indicate how rules should be used. Vampire proved seven of these lemmas and three more once some irrelevant axioms were removed. Only one lemma could not be proved at all, and we came close: only one of its seven subgoals could not be proved. Although this is a small sample, it suggests that Vampire indeed surpasses Isabelle's built-in tools in many situations.

Further experiments on those failed proof attempts were carried out. Among the six failed proof attempts on goals from `Message.thy`, three were made provable by removing some irrelevant axiom clauses.

Moreover, during the experiments with Isabelle/HOL's `Comb.thy` we also tried to translate HOL conjectures into FOL clauses without the inclusion of type information in order to compare the performance between the typed proofs and untyped proofs. These untyped experiments took the same set of goals (29 goals) from `Comb.thy`. Among these 29 goals, there were three Isabelle goals where Vampire found proofs more quickly when given untyped input and four goals where Vampire proved faster when given typed inputs. In particular, there was a case where Vampire proved a lot faster when given typed input (0.4 sec compared with 36 sec for untyped input). However, for those goals where untyped input required less time to be proved, the difference in time taken for typed and untyped input was not very significant. When typed input gave better performance, it should be explained by the restriction of search space. For the cases where untyped input outperformed typed input, it could be caused by the large literals in typed input due to the inclusion of types. Large literals may slow down proof search to some extent.

We also performed more specific experiments in order to examine whether the use of type information on overloaded operators can indeed cut down the search space. These experiments involved proving lemmas about subset relations taken from Isabelle/HOL's theory file `Set.thy`. In addition to the relevant axioms about subset properties, many irrelevant axioms about natural numbers were also included in the axiom set as they share the overloaded operator $\leq$. We first ran the experiments by not including the axioms of natural numbers and then ran the experiments again while adding those natural number axioms. Vampire spent the same amount of time in proofs regardless whether the natural number axioms were added or not. Clearly this demonstrates the benefits of including type

information of overloaded operators, since without these information, Vampire may pick up irrelevant natural number axioms in the proof search and hence would slow down the proof procedure.

Furthermore, several experiments on the formalisation of polymorphic equalities have been carried out. Among the goals from `Message.thy`, eight goals required the use of equality clauses (on sets), in either lemmas or conjectures. Five of them were proved by switching `SOS` off and were not proved otherwise. One of them was proved by turning `SOS` off and removing some irrelevant axioms. The other two could only be proved if the equality literals were replaced by two directional subset relations.

There were also some goals from `Multiset.thy` and `Message.thy` that were proved by applying results from several axiomatic type classes. This finding suggests that our formalisation of types and sorts is practical, while preventing the application of lemmas when the type in question does not belong to the necessary type class.

We have noticed that more proofs are found for Isabelle/HOL's lemmas than for Isabelle/ZF's lemmas. We believe that type information deserves the credit for this improvement: it reduces the search space.

5 Conclusion and Future Work

This paper has described the translation between Isabelle/HOL and first-order logic, with particular emphasis on the treatment of types. It has also reviewed some issues arising from our previous work involving Isabelle/ZF (an untyped formalism) and FOL.

The ZF transformation does not require an encoding of types: everything is represented by sets. Although it is based on FOL, there are many terms that are outside the scope of FOL, such as λ-terms, which need to be translated into FOL formulas.

Typed HOL formulas involve type and sort information, which must be preserved when translated into FOL. This paper has outlined the encoding of types and sorts in FOL. The use of such information should help automatic provers eliminate irrelevant axioms from consideration during a proof search. Moreover, it is essential in order to ensure soundness of proofs.

Experimental results for both ZF and HOL demonstrated the potential benefit of our integration. In particular, the results showed the encoding of HOL types and sorts was useful and practical. More experiments on compacting the encoding could improve Vampire's proof search speed as smaller literals should require less time for proofs.

There are some general issues that apply to both ZF and HOL. The treatment of equality is an example. Lemmas involving equality (typed or untyped) seem harder to prove and usually require us to turn off `SOS`. We tried to replace set equality by two subset relations before translating them into clauses, replacing $A = B$ by the conjunction of $A \subseteq B$ and $B \subseteq A$. We found that Vampire proved theorems much faster as a result. However, this approach is inflexible: proving a

pair of set inclusions is but one way of proving an equation. Therefore it would be harder to decide how we should replace equalities. Further investigation on this should be useful.

Our experimental results also gave us a hint on the proof strategy. Vampire's default settings are usually good, but sometimes proofs can only be found using other settings. We can either run several processes in parallel or run with the default settings first.

One aim of this integration is to relieve users of the task of identifying which of their previously proved lemmas are relevant. Our results show that existing proof procedures still do not cope well with large numbers of irrelevant axioms. More research on this issue is essential.

Soundness of the translation between different formalisms (for example HOL and FOL) will be ensured by various means. We intend to perform most of the translation inside Isabelle's logic rather than using raw code. We are modelling the types and sorts of HOL in full detail. We intend to execute the proofs found by the resolution prover within Isabelle.

Acknowledgements

We are grateful to the Vampire team (Alexandre Riazanov and Andrei Voronkov) for their co-operation and to Gernot Stenz and Geoff Sutcliffe for running some of our problems on their reasoning systems. Research was funded by the EPSRC grant GR/S57198/01 *Automation for Interactive Proof*.

References

1. Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integrating automated and interactive theorem proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction— A Basis for Applications*, volume II. Systems and Implementation Techniques, pages 97–116. Kluwer Academic Publishers, 1998.
2. Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automatic proof construction in type theory using resolution. *Journal of Automated Reasoning*, 29(3-4):253–275, 2002.
3. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.
4. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
5. Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics: TPHOLs '99*, LNCS 1690, pages 311–321. Springer, 1999.
6. Joe Hurd. An LCF-style interface between HOL and first-order logic. In Andrei Voronkov, editor, *Automated Deduction — CADE-18 International Conference*, LNAI 2392, pages 134–138. Springer, 2002.
7. Jia Meng. Integration of interactive and automatic provers. In Manuel Carro and Jesus Correas, editors, *Second CologNet Workshop on Implementation Technology*

for Computational Logic Systems, 2003. On the Internet at http://www.cl.cam.ac.uk/users/jm318/papers/integration.pdf.
8. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. LNCS Tutorial 2283.
9. Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Robinson and Voronkov [17], chapter 6, pages 335–367.
10. S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification: 8th International Conference, CAV '96*, LNCS 1102, pages 411–414. Springer, 1996.
11. Lawrence C. Paulson. Generic automatic proof tools. In Robert Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, chapter 3. MIT Press, 1997.
12. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
13. Lawrence C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3):73–87, 1999.
14. Lawrence C. Paulson. Isabelle's isabelle's logics: FOL and ZF. Technical report, Computer Laboratory, University of Cambridge, 2003. On the Internet at http://isabelle.in.tum.de/dist/Isabelle2003/doc/logics-ZF.pdf.
15. A. Riazanov and A. Voronkov. Efficient checking of term ordering constraints. Preprint CSPP-21, Department of Computer Science, University of Manchester, February 2003.
16. Alexander Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning — First International Joint Conference, IJCAR 2001*, LNAI 2083, pages 376–380. Springer, 2001.
17. Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science, 2001.
18. Jörg Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, Immanuel Normann, and Martin Pollet. Proof development with ωmega: The irrationality of $\sqrt{2}$. In Fairouz Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, pages 271–314. Kluwer Academic Publishers, 2003.
19. Geoff Sutcliffe and Christian Suttner. The TPTP problem library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, October 1998.
20. Christoph Weidenbach. Combining superposition, sorts and splitting. In Robinson and Voronkov [17], chapter 27, pages 1965–2013.
21. Markus Wenzel. Type classes and overloading in higher-order logic. In Elsa L. Gunter and Amy Felty, editors, *Theorem Proving in Higher Order Logics: TPHOLs '97*, LNCS 1275, pages 307–322. Springer, 1997.