Integration of Interactive and Automatic Provers

Jia Meng

Computer Laboratory, University of Cambridge
Jia.Meng@cl.cam.ac.uk

Abstract. Interactive and resolution based automatic provers have both been used widely. Interactive provers offer users expressive formalisms and flexibility and are suitable for proving theorems of any user defined logics. However, they provide limited automation. In comparison, resolution based automatic provers provide automation, but only allow first order logic with equality. I am investigating combining the two types of provers through integrating an interactive prover Isabelle and a resolution based automatic prover Vampire. This paper gives an overview of the integration and describes the techniques used in it. It also lists some experimental results.

1 Introduction

Interactive theorem provers support expressive formalisms which allow users to define functions, recursive types and embed complicated logics and theories. However users must guide each step of proofs and hence require significant human efforts. Resolution based provers are automatic but only allow first order logic with equality. We investigate combining the interactive provers' flexibility and resolution provers' automation through integration.

In this paper I describe the progress of integrating a generic interactive theorem prover Isabelle [6] and a resolution prover Vampire [8].

1.1 Isabelle and Vampire

Isabelle is a powerful interactive theorem prover based on the typed λ-calculus. It is generic: users can embed various logics inside. Currently the supported logics include HOL (Higher Order Logic), HOLCF (the definitional extension of Church's Higher-Order Logic with Scott's Logic for Computable Functions), FOL (First Order Logic), ZF (ZF Set Theory based on FOL).

A proof goal in Isabelle is a statement expressed in some logic. Users need to specify what tactics and rules to use in order to prove a goal or decompose it into several smaller subgoals. A rule is a previously proved lemma or theorem. A tactic tells Isabelle how the rules should be applied. There are several built-in classical reasoning tactics, such as `blast` and `fast` (based on tableau methods) as well as equality reasoning tactics, such as `auto` and `simp` (based on equality rewriting). They take on a default set of classical rules and equalities in the current context of the goal and attempt to find a proof automatically. However, users still need to choose the right tactics and this requires proof skills.

Vampire is one of the leading resolution theorem provers for first-order logic with equality. Its paramodulation rule deals with equality. In each of the past three years it has won some category of the CADE ATP System Competition.

1.2 Aim of This Integration and Its Novelties

The aim [7] of integrating Isabelle and Vampire is that when a user attempts to prove an Isabelle goal, instead of asking the user to specify what tactics or rules to use, the goal will be translated to first-order logic clauses along with a suitable set of rules. These rules are taken from the default configuration of `blast` and `auto`. Isabelle's existing provers use these rules already since they are already-proved theorems. These clauses will then be sent to Vampire. Vampire will run in the background, attempting to prove each unsolved subgoal. When a proof is found, the proof will be sent back to Isabelle for verification. Vampire will combine the classical and equality reasoning through resolution and paramodulation, and will replace many Isabelle's tools such as `blast` and `auto`. Furthermore, our integration also aims to improve the automation by automatically proving goals that cannot be proved by the Isabelle's built-in tools.

There have been several previous attempts at integrating interactive and automatic theorem provers. KIV has been integrated with ${}_3T^AP$ by Ahrendt et al. [1]. We are hoping to improve upon their results through the use of Vampire, which is one of the world's best automatic provers. The HOL system [2] has been integrated with a model elimination prover developed by John Harrison [3] and was later integrated with prover Gandalf by Joe Hurd [4]. However, the users must collect relevant lemmas manually.

Two major novelties of our integration are:

- Users are less likely to have to collect lemmas manually. They do not have to specify proof tactics either.
- As Vampire will run in the background, users will not have to wait for a response from Vampire before continuing with the rest of the proof.

I have experimented the link up between Isabelle/ZF (with equality) and Vampire. Isabelle/ZF is based on first order set theory and is untyped. The rest of this paper discusses the following aspects of the work involved:

1. Translation between Isabelle/ZF formalism and first-order logic.
2. Minimisation of Vampire proof search space.
3. Vampire's weight and precedence assignments to function and predicate symbols.
4. Settings of Vampire and their influence to proof performance.
5. A collection of results.
6. Conclusion and future work.

2 Translation Between Isabelle/ZF and First-Order Logic

A practical way of translating between the two formalisms is essential for efficient proving by Vampire. Therefore I have carried out many experiments which consisted of taking `blast`, `fast`, `clarify`, `auto`, `simp` invocations from existing proofs and attempting to reproduce proofs using Vampire. During a proof, a set of classical rules and equality rewriting rules in the current Isabelle context are translated to first-order axiom clauses. The goals are negated, converted to clauses and sent to Vampire.

As it was experimentation, the translation between the formalisms was done manually.

Isabelle/ZF formulas (rules and goals) that are already in first-order logic (FOL) form can be translated directly to clauses using Clause Normal Form (CNF) transformation. For other Isabelle/ZF formulas that are not expressed in FOL form, reformulation of the formulas is required before CNF transformation.

2.1 Transformation of Formulas Not in FOL Form

Isabelle/ZF rules, such as the elimination rule for set intersection (`IntE`),

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

need to be translated into a FOL formula ($\cap E$),

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

since the predicate variable P is not allowed in FOL. This translation is correct: the first formula is the elimination rule written in the natural deduction form and we need to unfold it in order to eliminate the predicate variable P. Although the predicate P disappears, we are not losing information during a proof. Suppose we are going to prove a goal

$$[(i \in S_1 \cap S_2) \wedge C_1 \wedge C_2 \ldots \wedge C_n] \rightarrow Q$$

where $C_1 \ldots C_n$ and Q are formulas. In Isabelle, this goal is proved by applying the set intersection elimination rule `IntE`. Firstly $c \in A \cap B$ is instantiated to $i \in S_1 \cap S_2$, and P to Q (c, A, B, P are universally quantified variables). Then the goal is replaced by a new subgoal

$$[C_1 \wedge C_2 \ldots \wedge C_n \wedge i \in S_1 \wedge i \in S_2] \rightarrow Q$$

The negation of this subgoal generates the same set of clauses as what we will have if we resolve the negation of the original goal with the formula $\cap E$ above.

Moreover, rules such as `domainE`,

$$\forall a\, r\, P\, [(a \in domain(r) \wedge \forall y\, (\langle a, y\rangle \in r \rightarrow P)) \rightarrow P]$$

should be translated into

$$\forall a\, r\, [a \in domain(r) \rightarrow \exists y\, (\langle a, y\rangle \in r)]$$

We also need to remove Isabelle's terms, such as $\bigcap_{x\in A} B(x)$, since they are not present in first-order logic. A term $y \in \bigcap_{x\in A} B(x)$ is translated into

$$\forall x\, [x \in A \rightarrow y \in B(x)] \wedge \exists a\, [a \in A]$$

In addition, a formula $\phi(\bigcap_{x\in A} B(x))$ is equivalent to $\exists v\, [\phi(v) \wedge \forall u\, (u \in v \leftrightarrow u \in \bigcap_{x\in A} B(x))]$.

2.2 Some Other Issues of Translation

A subset relation $R \subseteq S$ is equivalent to $\forall x\, (x \in R \rightarrow x \in S)$ (this reduces the subset relation to the membership relation).

Experiments show 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. The experimental results are shown in section 5.1

Some formulas involve set equality as $A = B$. During many proofs set equality predicates should be reduced to two subset predicates by resolution: $A \subseteq B$ and $B \subseteq A$. However, Vampire usually gives the positive equality literal (`equal`) a low priority relative to other literals in the same clause. Therefore the positive equality literal is likely to be selected and resolved last during resolution. Experiments show that better performance can be achieved when set equality predicates are replaced by the subset predicates, which will be further reduced to formulas involving membership predicates as shown above. The experimental results are shown in section 5.2.

2.3 Minimising Search Space

The size of the search space (in terms of the number of clauses) plays a significant role in the performance of resolution provers. Formula renaming [5] has been used to reduce the number of clauses generated during CNF transformation.

For instance, the rule `Ap_contractE` generates 135 clauses using standard CNF transformation. After applying formula renaming to this rule, the CNF transformation generates 15 clauses.

In the attempt of proof reproduction in Vampire, four proof goals required the use of `Ap_contractE`. None of the four goals were proved when standard CNF transformation was used. However, three of them were proved after applying the formula renaming method.

3 Weight, Precedence Assignment

During a resolution proof, if we want a literal to be eliminated sooner, then it should have a higher weight relative to other literals in the same clause. This difference in weights gives an ordering on literals. Vampire uses Knuth-Bendix Ordering (KBO) [9] to compute this ordering on literals. Furthermore, KBO is parameterized by weights and precedences of functions and predicates, which can be assigned explicitly by users. Correct weight, precedence assignment is important for several reasons.

An Isabelle proof usually requires definitions of constants, functions, etc. to be unfolded before tactics can be applied. However, these definitions can be sent to Vampire as equality clauses, so that users do not have to specify which definition should be unfolded. In order to have Vampire replace the definiendum by the definiens through ordered paramodulation, we need to assign greater weights to functions that occur in the definiendum.

An example was a proof of lemma `I_contract_E` in an Isabelle/ZF theory file `Comb.thy`. It proves that combinator I (the identity combinator) has no possible contraction. An axiom clause of definition $I = KSS$ was included in the Vampire axiom set. Without assigning a higher weight to I relative to K and S, ordered paramodulation will never replace I by KSS as the latter is much heavier than the former (all function symbols have the same weight by default). After the assignment was done, Vampire proved the goal quickly.

More importantly, Isabelle rules have information indicating whether a rule should be used as an elimination rule (forward chaining) or an introduction rule (backward chaining). This information is lost after the rules are translated into clauses. Weight and precedence assignment in Vampire is probably the only way to preserve this information. However the resulting KBO is a partial ordering on terms with variables. Therefore in some situations, it is possible that weight assignment will not produce any effect.

4 Settings of Vampire

Vampire allows users to specify various settings of the prover. Considerable experimentation with numerous settings of Vampire was carried out in order to find the best combinations of these settings. This was done by attempting to prove around 250 lemmas taken from Isabelle/ZF's theory files `equalities.thy` and `Comb.thy`. Each lemma presents between 1 and 4 separate goals.

The experiments show that the default setting of Vampire is usually good. Moreover, the literal selection mode is the most important factor in determining the speed of proofs. Four selection modes (`selection4`, `selection5`, `selection6` and `selection7`) are better than the others. Vampire also supports Set of Support Strategy (SOS). Most of the goals require us to turn on SOS in order to find proofs.

The experiments show that five combinations of settings are better. They were written to five separate setting files so that five Vampire processes can run in parallel. The result of some tests is shown next.

5 Results

5.1 Comparisons of Subset and Membership Relations

In order to investigate whether we should reduce subset relations to the equivalent membership relations, proofs of 32 goals were carried out. These goals involve either positive subset predicates or negative subset predicates, or both. Without replacing any subset predicate, only 17 goals were proved. However, after I removed all subset predicates 30 goals were proved. A more detailed comparison is shown in Table 1 below.

Table 1. Number of Goals Proved with and without Subset Relations

	Positive Subset	*Negative Subset*	*Both*
Number of Goals Involving	14	31	13
Number of Goals Proved if only Keeping	11	18	6
Number of Goals Proved if Removing Both Positive and Negative Subset	13	30	13

Some explanation of Table 1 may be useful. The intersection of row `Number of Goals Involving` and column `Positive Subset` indicates the total number of goals where positive subset predicates exist. These goals may involve negative subset predicates as well. The intersection of row `Number of Goals Proved if Removing Both Positive and Negative Subset` and `Positive Subset` indicates that 13 goals were proved (out of 14 goals that involve positive predicates) once all subset predicates (both positive and negative) were removed. Similarly for other columns and rows.

5.2 Tests on Equality Literals

I tried to prove twelve goals involving equality literals. Eleven of these goals have negative equalities and one has a unit clause with a positive equality literal. Vampire quickly found a proof for the goal containing the positive equality, as I had expected: in a unit clause, a positive equality is definitely selected and resolved with some negative equality literal (a negative equality literal receives a higher weight than other literals occurring in the same clause). In comparison, only one goal involving a negative equality was proved. Once I removed all negative equalities using subset and then membership literals, 10 goals were proved.

5.3 First Comparison of Isabelle/ZF and Isabelle/ZF-Vampire

The aim of the first comparison was to find out how well can Isabelle/ZF-Vampire integration prove goals that were originally proved by Isabelle's built-in tools, such as `blast`, `auto`, `simp`, `clarify` etc. This experiment is also important to demonstrate whether the integration can improve automation by combining the built-in tools, which need to be performed separately in Isabelle.

The initial set of tests was proving around 250 lemmas. They were taken from `equalities.thy` and `Comb.thy`. Around 70 axiom clauses were included in the axiom set. Most of the goals were proved with this axiom set.

As the ultimate aim is to give Vampire a large set of axioms, tests with a larger axiom set were necessary. During this second run of tests around 129 to 160 axioms clauses were used (As it was explained in section 1, these axiom clauses are generated by the rules taken from the default configuration of `blast` and `auto`.). Vampire tried to prove 37 lemmas (63 separate goals), which were drawn from the previous 250 lemmas. Each lemma was attempted five times using the five combinations of settings: `defaultSetting` uses default settings; `setting1` to `setting3` use Vampire's literal selection mode selection5 to selection7 respectively; `setting4` turns on dynamic splitting. The lemmas from `equalities.thy` were mainly proved by `blast` tactic (with other tactics such as `clarify`, `simp` as well), while lemmas from `Comb.thy` are more complicated and many of them also used `auto` tactic. The results are shown in Table 2 below.

The time limit for each attempt of proof was 60sec.

Table 2. Number of Goals Proved with the Large Axiom Set

Setting File	*Number of Goals Proved from* `Comb.thy`	*Number of Goals Proved from* `equalities.thy`	*Total Number of Goals Proved*
defaultSetting	21	28	49
setting1	22	28	50
setting2	21	27	48
setting3	21	28	49
setting4	18	27	45
Combination of all settings	24	28	52

Fifty-two goals out of 63 were proved by the combination of all five settings within the time limit. Many tactics, such as `blast`, `auto`, can indeed be replaced by Vampire.

The tests also indicate a potential that more goals can be proved by running several Vampire processes with different settings in parallel, although the results so far are not dramatic. In addition, there are 8 relatively complex goals where the amount of time taken by each setting to find proofs varies significantly. It shows that goals could be proved more quickly by running several Vampire processes in parallel. Performance variance in different settings is more significant when proving more complicated lemmas (here lemmas drawn from `Comb.thy`).

5.4 Second Comparison of Isabelle/ZF and Isabelle/ZF-Vampire

A more important aim of this integration is to automatically prove goals that cannot be proved by Isabelle's built-in tools and hence reduce user interaction. This second comparison examined whether the integration can prove goals that were not proved by `blast`, `auto`, `simp` etc. Isabelle proofs of these goals consist of several proof steps carried out by users. If these goals can be proved automatically by Vampire, then Isabelle users will not have to specify the proof steps. This set of experiments took 15 lemmas from Isabelle/ZF theory files `Comb.thy` and `PropLog.thy`.

One point that we need to consider is at which stage of a proof, we should send the current goal/subgoals to Vampire for an automatic proof. Induction is sometimes necessary to prove a goal and we are not aiming to automate this induction step. Therefore for those lemmas that were proved by induction in Isabelle, I sent to Vampire those subgoals I was left with after induction was performed.

The results are shown in Table 3. Some lemmas present more than one subgoal to Vampire. Ten lemmas were proved by Vampire. For those lemmas that could not be proved by Vampire automatically, one or more subgoals' proofs were not found. The combination of the five Vampire settings was used during the tests. The lemma IDs marked with asterisks indicate that I attempted to eliminate all Isabelle proof steps for these lemmas.

Table 3. Number of More Complex Goals Proved by Vampire

Lemma ID	*Number of Isabelle Proof Steps Eliminated*	*Number of Subgoals Sent to Vampire*	*Number of Subgoals Proved by Vampire*
1	4	2	2
2	2	2	1
3	3	2	2
4	2	1	1
5*	2	1	1
6*	4	1	1
7*	2	1	1
8*	3	1	1
9*	4	1	1
10*	6	1	1
11	3	3	1
12*	3	1	0
13*	2	1	0
14	10	2	1
15	4	2	2

6 Conclusion and Future Work

The current experimentation clearly demonstrates the potential of integrating Isabelle and Vampire. User interaction with Isabelle during a proof is significantly reduced. We will next implement the automatic communication between the two provers. This implementation will consist of several parts:

- Automatic translation of formulas between Isabelle/ZF and Vampire's First Order Logic. This should not be difficult once we have found a practical way for this translation.
- Implementing an interface between the two provers. At some point during an Isabelle proof, a goal or some subgoals may be sent to the automatic provers. One or more automatic provers may sit at some servers over the network, running several processes (with different settings) in parallel and attempt to solve any unsolved subgoal. Since it might take a while for the automatic provers to find a proof, users should be able to carry on with the rest of the proof. The way that these automatic and interactive provers interact may be important to the overall performance of the integration. As there is much communication between the provers, we need to ensure the communication can be carried out quickly and efficiently.
- Proof reconstruction in Isabelle. Once a proof is found by Vampire, we need to send the proof back to Isabelle for verification. However, resolution based provers' proofs are hard to read, especially if Skolemization is used. We need to implement such proof reconstruction efficiently in Isabelle.

Furthermore, future work will also include investigation on how to use Vampire's weight and precedence assignments more effectively.

Trials on other Vampire's settings will help fine tune Vampire's performance.

In addition, integration of other supported logics of Isabelle with Vampire will take place.

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 of *Applied Logic Series, No. 9*, pages 97–116. Kluwer, Dordrecht, 1998.
2. M. J. C. Gordon and T. F. Melham. *Introduction to HOL (A theorem-proving environment for higher order logic).* Cambridge University Press, 1993.
3. John Harrison. A mizar mode for HOL. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220, Turku, Finland, 1996. Springer-Verlag.
4. 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, 12th International Conference, TPHOLs '99*, volume 1690 of

Lecture Notes in Computer Science, pages 311–321, Nice, France, September 1999. Springer.

5. Andreas Nonnengart and Christoph Weidenbach. Computing Small Clause Normal Forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 6, pages 335 – 367. Elsevier, Amsterdam, Netherlands, 2001.
6. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994. LNCS 828.
7. Lawrence C Paulson. Automation for interactive proof. Grant Proposal to the U.K. Engineering and Physical Sciences Research Council, 2002.
8. A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.
9. A. Riazanov and A. Voronkov. Efficient checking of term ordering constraints. Preprint CSPP-21, Department of Computer Science, University of Manchester, February 2003.