Experiments with ATP Integration in a Software Engineering Application

Thomas Baar[1], Bernd Fischer[2], and Dirk Fuchs[3]

[1] Inst. f. Mathematik, HU Berlin
baar@mathematik.hu-berlin.de
[2] Abt. Softwaretechnologie, TU Braunschweig
fisch@ips.cs.tu-bs.de
[3] Fachbereich Informatik, U Kaiserslautern
dfuchs@informatik.uni-kl.de

Abstract. We describe a combination of the NORA/HAMMR software component retrieval tool and the ILF system which provides the necessary infrastructure to apply different first-order theorem provers to the emerging proof problems. This framework allows the cooperation of independent deductive subsystems in two different modes. Our results show that both modes—competition between problem variants or provers and proper cooperation following the TECHS approach—improve the success rate considerably.

1 Introduction

In this paper, we report on work in the NORA/HAMMR-project to integrate and combine different ATPs for a specific software engineering application, deduction-based software component retrieval. In a nutshell (cf. [FSS98] for a more detailed account), deductive retrieval uses formal specifications as indexes and queries, builds proof tasks from these, and checks the validity of the tasks using an ATP. A component is retrieved if the prover succeeds on the associated task—retrieval becomes a deductive problem.

As earlier experience shows, no ATP yields acceptable results if the proof problems are translated naïvely from the application logics (given by a formal specification framework) into its input language. Thus, NORA/HAMMR provides several mechanisms to control this transformation process and to simplify the proof tasks. Here, we investigate the question whether one of the possible (semantic equivalent) forms of a proof task is the best one for ATPs.

Experience also shows that the proof-times of ATPs are notoriously difficult to predict and may vary eratically for "similar" problems. Moreover, the different ATPs deliver their best results on different subsets of the emerging proof tasks which makes it hard to pick the "right" prover. In NORA/HAMMR, we turn this difficulty into an advantage and use the ATPs in competition to improve the overall results of retrieval. Further improvement can then be achieved by a proper cooperation of ATPs. However, both approaches require a complicated infrastructure (e.g., for scheduling, information exchange, and control) which is

not be provided by NORA/HAMMR. Instead, we use the ILF tool [DGHW97] which was specifically developed for such tasks.

2 Experimental Settings

Deduction-based component retrieval [MW95,PBA95,MMM97,FSS98] exploits semantic information to locate software components, e.g., PASCAL-procedures, in a library. The components are described exactly by *contracts*—formal specifications of their pre- and postconditions. Queries are specified in the same way. A query q is implemented by a candidate c if the following theorem holds:

$$(pre_q \Rightarrow pre_c) \wedge (pre_q \wedge post_c \Rightarrow post_q)$$

As we have argued in [FSS98], contracts should not be formulated in pure FOL but in a richer "custom logics", e.g., VDM-SL. A first integration step removes these custom constructs, e.g., *let*-expressions or pattern matching. In NORA/HAMMR, this is done via LPF (Logic of Partial Functions) as intermediate layer. A second integration step translates the three-valued LPF into FOL, using a standard algorithm [JM94]. However, utmost care must be taken to control the size of the resulting formulas.

Since none of the ATPs we apply is a proper inductive prover, this step also approximates inductively defined sorts (e.g., lists). It encodes the free generation property by additional first-order axioms, i.e., it encodes (*i*) the constructor property of the constructor functions (i.e. that terms with different top-level constructors are never equal), (*ii*) the surjectivity of the constructors wrt. to the data type domain (i.e. that the top-level function symbol of each element in the domain is one of the constructor functions), and (*iii*) the freeness or injectivity of the constructor functions (i.e. if two terms with the same top-level constructor are equal then their respective arguments are equal, too.) For example, in the usual theory of lists which is freely generated by *nil* and *cons*, the three properties give rise to the following axioms (*i*) $\forall i : item,\ l : list \cdot nil \neq cons(i, l)$, (*ii*) $\forall l : list \cdot l = nil \vee \exists i : item,\ m : list \cdot l = cons(i, m)$, and (*iii*) $\forall i, j : item,\ l, m : list \cdot cons(i, l) = cons(j, m) \Rightarrow i = j \wedge l = m$. Obviously, the induction scheme which follows from a data type definition cannot be encoded by first-order axioms. However, the special nature of our proof tasks allows the powerful heuristic to use the formal parameter(s) of candidate component as induction variable(s) and to instantiate the induction scheme appropriately.

Since the proof tasks are automatically generated, they often contain unnecessary constructs and thus allow rigorous simplification. In NORA/HAMMR, we use rewrite-based simplifications which eliminate propositional constants, rewrite into conjunctive normal form and then further into anti-prenex form to minimize the quantifier scopes. Some of the rules are domain specific, e.g., a lemma `memNil` induces a rule $mem(x, nil) \rightsquigarrow false$. The constructor properties as injectivity and surjectivity induce other rules like $\exists x : List \cdot x = t \rightsquigarrow true$. If it is possible to rewrite a proof task to *true* (*false*), simplification can within NORA/HAMMR also be regarded as a confirmation (rejection) filter.

In our experiments, the simplified tasks were proved in a theory over lists which is organized in several subtheories introducing axioms and/or lemmas. The large number of usable formulas contained in such a theory database requires a reduction mechanism which selects only those which are necessary to find a proof at all or are likely to shorten it and omits all those which only increase the search space.

In NORA/HAMMR, we use signature-based heuristics similar to that of Reif and Schellhorn [RS98]. Their basic assumption is that rules are redundant if they contain no symbols which occur in the problem, or more precisely, if they are defined in redundant theories. A theory is redundant if it introduces only symbols not occurring in the problem and is not referred (directly or indirectly) by other non-redundant theories. NORA/HAMMR implements this selection mechanism and provides two selection strategies: (*i*) select only axioms, (*ii*) select all axioms and lemmas from non-redundant theories.

3 Competition Experiments

With competition we denote that the ATPs work in parallel on basically the same problem but do not exchange information. We can distinguish two different competition modes which work along independent dimensions:

- variant competition: multiple identical instances of a single prover work on different task formulations of a problem.
- system competition: different provers or different instances of a prover (e.g., using different strategies or control parameters) work on the same proof task.

In NORA/HAMMR, we have experimented with both competition modes. For these experiments, we used a library comprising 119 specifications of list processing functions and cross-matched each specification against the entire library. This yielded 14161 proof tasks where 1839 or 13.0% were valid.

3.1 Variant competition using SPASS

The theory database used in the experiments comprises 65 theories, in which 24 different function and predicate symbols are axiomatized. The axiomatization consists of 38 core axioms and approximately 100 additional lemmas which are (first-order or inductive) consequences of the axioms.

The different axiom selection mechanisms give rise to quite different search spaces. To exploit these, we used the techniques described above and generated different axiom sets for each problem where *core* and *lemmas* denote the selection of axioms only and axioms and lemmas from non-redundant theories, respectively, while the *full* set contains the entire theory database (cf. Table 1).

We then used the SPASS [WGR96] prover to solve the three sets of proof tasks. Table 1 shows the results for different timeouts.[1] As expected, the smaller

[1] All results were obtained using SPASS V0.80 on a 200MHz PentiumPC with 64MB running Linux.

$T_{\max}$ (secs.)	core	lemmas	full	comp cl	comp cf	comp lf	comp all
1	1089	969	933	1128	1124	973	1129
10	1200	1201	1190	1284	1299	1243	1321
30	1236	1235	1280	1329	1379	1317	1390
60	1250	1258	1321	1346	1413	1356	1420

Table 1. Results of variant competition experiments

search spaces induced by the *core* selection mechanism lead to a significantly (approx. 15%) higher number of fast proofs which is especially important for our application. Surprisingly, however, and in contrast to the observations of [RS98], none of the heuristics pays in the long run: for timeouts greater than 10 secs., SPASS was able to solve more problems when *redundant* axioms and lemmas were added. We conjecture that some of them describe properties of, e.g., the *hd*- and *tl*-functions which indirectly also apply to, e.g., the *append*-function but are not explicitly formulated as lemmas for *append*.

However, these problems do not invalidate the entire selection mechanism: as expected, competition between the different variants significantly increases the number of proofs found. The benefits vary with the timeout and the selected variants and reach a maximum of 12.5% compared to the best single variant and an overall increase of 7.5% for a timeout of 60 secs. and full competition between all variants. For timeouts shorter than 20 secs. we can even observe a "superlinear" increase. E.g., for a timeout of 10 secs., competition between all three variants solves 3.2% more problems than the best variant with a timeout of 30 secs. At the same time, the total elapsed runtime drops by approx. 6%.

3.2 System competition controlled by ILF

Within NORA/HAMMR the ILF-system can be used as a shell for ATPs. ILF launches several ATPs on different machines of a local network at the same time.

The experiments for system competition are based on a representative subset of the original library. We selected 24 components; in the resulting 576 tasks, the preprocessing methods integrated in NORA/HAMMR identified 23 provable and 336 unprovable tasks, which can be simplified to *true* and *false*, respectively.

For the remaining 217 tasks, the provers OTTER, SPASS, and SETHEO were started, each with a timeout of 120 secs. For SPASS and SETHEO we used different type-encoding techniques provided by ILF. Such techniques are needed to transform formulas from sorted logic into an unsorted logic.

Table 2 contains the results; SPASS_{te} and $\text{SETHEO}_{\text{te}}$ denote variants were a simple term-encoding technique was used (this was also used for OTTER), $\text{SPASS}_{\text{rel}}$ denotes a variant with standard predicate relativization technique, and $\text{SETHEO}_{\text{sub}}$ a more complicated term-encoding technique where the type-subtype relation is coded by a term-instance relation on the codeterms for types. Each prover (variant) is compared with the every other one. For instance, the

	OTTER	$\text{SETHEO}_{\text{sub}}$	$\text{SETHEO}_{\text{te}}$	$\text{SPASS}_{\text{rel}}$	SPASS_{te}	comp
OTTER	**46**	25	14	4	12	–
$\text{SETHEO}_{\text{sub}}$	3	**24**	6	1	3	–
$\text{SETHEO}_{\text{te}}$	9	23	**41**	3	6	–
$\text{SPASS}_{\text{rel}}$	21	40	25	**63**	17	–
SPASS_{te}	13	26	12	1	**47**	–
comp	–	–	–	–	–	**70**

Table 2. Results for provable tasks within ILF

first row shows that OTTER solved 46 proof tasks, and of these 25 could not be solved by $\text{SETHEO}_{\text{sub}}$, 14 not by $\text{SETHEO}_{\text{te}}$, 4 not by $\text{SPASS}_{\text{rel}}$, and 12 not by SPASS_{te}. If all provers are run competetively, a total of 70 tasks can be solved, i.e., compared to the results of the best ATP, the recall rate can be increased significantly by 11%. As a further remarkable point, we observe that no prover variant is "subsumed" by another variant. Even for SPASS which has built-in support for sorts, the term encoding yields an additional proof.

However, in order to show formally for every case, whether the query matches or not, the remaining 147 tasks have to be shown unprovable. We thus started all provers on the negated goals and obtained even better results as in the "affirmative" case–competition can solve 56% more tasks than the best single system.

	OTTER	$\text{SETHEO}_{\text{sub}}$	$\text{SETHEO}_{\text{te}}$	$\text{SPASS}_{\text{rel}}$	SPASS_{te}	comp
OTTER	**41**	31	27	16	24	–
$\text{SETHEO}_{\text{sub}}$	7	**17**	7	9	12	–
$\text{SETHEO}_{\text{te}}$	13	17	**27**	12	14	–
$\text{SPASS}_{\text{rel}}$	14	31	24	**39**	14	–
SPASS_{te}	9	21	13	1	**26**	–
comp	–	–	–	–	–	**64**

Table 3. Results for unprovable tasks within ILF

4 Cooperation Experiments

The ILF system allows—as already mentioned—for a proper cooperation of different theorem provers by using the TECHS approach [FD97,DF98].

TECHS requires several different provers running in parallel on different computing nodes. All provers tackle the same proof problem (which is given to the

provers in an initialization phase) independently during working phases. The general idea of the TECHS approach is to achieve cooperation between these provers by periodically interchanging selected clauses in cooperation phases. The selection of clauses is performed by so-called *send-* and *receive-referees*. These referees allow for a *success-driven* and *demand-driven* exchange of clauses between different provers. Note that a team based on the TECHS approach can easily be integrated into the ILF system. It is only necessary that ILF launches the provers and gives them information on the proof problem and their cooperation partners. After that, the provers can cooperate independently of ILF.

For our experimental study regarding cooperative provers we restricted ourselves so far to the provers SPASS and SETHEO. More exactly, we even restricted the direction of the information exchange and allowed only SPASS to give some clauses to SETHEO. Indeed, SETHEO is able to produce lemmas which can be given to SPASS. However, since these lemmas are—due to instantiations needed to close tableau branches not needed for deriving the lemmas—not as general as they could be. Hence, they usually cannot be used in contracting inferences which are the most important inferences of a saturation-based prover like SPASS (w.r.t. performance) and do not entail much gain. Thus, our cooperating team consists of the prover SPASS which essentially works as sequentially and one or more instances of SETHEO (see below) which process the lemmas from SPASS.

Cooperative runs were performed as follows. In each initialization phase both provers obtained as initial clause set the results of SPASS' normal form translator FLOTTER. Since SETHEO is not able to utilize built-ins for equality the usual equality axioms were added to its clause set. Note that—when using SETHEO in ILF—it usually obtains clauses from a different normal form translator. It turned out, however, that for cooperation purposes the use of identical normal forms is unavoidable. If the provers work on different kinds of normal forms (including different signatures due to different Skolemization procedures) SETHEO is in general not able to close many tableau branches with the help of SPASS lemmas because unification failures arise immediately. It is to be emphasized that SETHEO's performance (when working alone) was not weakened when employing the FLOTTER normal form.

In the working phases we let the provers work with following options. SPASS employed its standard setting. However, we did not allow SPASS to use its *splitting rule*. This rule realizes some kind of case analysis entailing that the prover has to work with semantically invalid clauses during the proof run. This is no problem for SPASS since it manages the dependencies between different subproblems. But such clauses cannot be given to SETHEO. Because of the fact that in our experiments SPASS very often used the splitting rule and hence did only produce very few valid (unit) clauses which could be given to SETHEO, we did not use the splitting rule. Note that for our test set the performance of SPASS was identical regardless whether or not it used splitting. For SETHEO we chose options which were automatically generated as described in [MIL+97]. Specifically, the weighted-depth bound [MIL+97] was automatically chosen. We experimented additionally with the depth bound ([LMG94]).

In a cooperation phase at most 35 units were transferred to an instance of SETHEO. Experiments with non-unit clauses did not lead to better results. We employed a fixed setting for the referee parameters (see [FD97]).

We performed our first experimental studies in the light of the same problems as in Section 3.2. All in all, we tackled 81 provable problems. Results can be found in Table 4. Results of SPASS, SETHEO using the weighted depth bound (SETHEO wd), and SETHEO using the depth bound (SETHEO d) are displayed in columns 2–4. Columns 5–7 present results of our cooperating system. Columns 5 and 6 display runtimes of a 2-prover team consisting of SPASS and SETHEO using the weighted depth (coop wd) and the depth bound (coop d), respectively. Column 7 shows the results of a 3-prover team consisting of SPASS, SETHEO using the weighted depth bound, and SETHEO using the depth bound (coop all). Finally, column 8 gives the results of an analogous competitive 3-prover team.

solved	SPASS	SETHEO wd	SETHEO d	coop wd	coop d	coop all	comp
≤ 10	37	39	37	49	48	50	48
≤ 30	47	39	40	57	56	58	54
≤ 60	48	45	40	57	57	58	55
≤ 120	50	48	40	60	58	61	56

Table 4. Experiments with cooperating theorem provers

Table 4 reveals the high potential of cooperation. The number of solved problems could be increased, additionally the runtimes could be decreased. It is to be expected that the results can further be improved. Firstly, [Fu98] shows a way of how to extract information from SETHEO which might improve the performance of SPASS. Secondly, increasing the team of cooperating provers, e.g. by additionally using DISCOUNT, may lead to a further gain of efficiency.

5 Conclusions

In this paper, we reported on the practical advantages we gained from the integration of several deductive methods into NORA/HAMMR.

It turns out that reasonable simplification during the task generation is necessary in order to get acceptable results. Furthermore the complexity of the usable theory requires a selection of axioms. Proper theorem selection is a generally problem in automated deduction, in our experiments the best results for SPASS can be achieved by a competitive run of core- and full- selection strategy. Even greater is the advantage of ATP-competition compared with each single prover. Here we have an improvement till 56% for some kinds of proof tasks.

Not only the combination of ATPs in a competitive way, also the proper cooperation of ATPs can increase the success rate in some cases.

References

[BS98] W. Bibel and P. H. Schmitt, (eds.). *Automated Deduction - A Basis for Applications.* Kluwer, Dordrecht, 1998. To Appear.

[DF98] J. Denzinger and D. Fuchs. Enhancing conventional search systems with multi-agent techniques: a case study. In *Proc. Int. Conf. on Multi Agent Systems (ICMAS) 98*, Paris, France, 1998. To Appear.

[DGHW97] B. I. Dahn, J. Gehne, Th. Honigmann, A. Wolf. "Integration of Automated and Interactive Theorem Proving in ILF". In Proc. CADE-14, LNAI 1249, pp. 57-60, Springer, 1997.

[FD97] D. Fuchs and J. Denzinger. Knowledge-based cooperation between theorem provers by TECHS. Technical Report SR-97-11, U. Kaiserslautern, 1997.

[Fu98] D. Fuchs. Cooperation between Top-Down and Bottom-Up Theorem Provers by Subgoal Clause Transfer. In *Proc. 4th Int. Conf. on Artificial Intelligence and Symbolic Computation (AISC)*, Plattsburgh, NY, USA, 1998. To Appear.

[FSS98] B. Fischer, J. M. P. Schumann, and G. Snelting. "Deduction-Based Software Component Retrieval". In Bibel and Schmitt [BS98]. To Appear.
Proc. Aspen
Proc.

[JM94] C. B. Jones and K. Middelburg. "A Typed Logic of Partial Functions Reconstructed Classically". *Acta Informatica*, **31**(5):399–430, 1994.

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *J. Automated Reasoning*, **13**:297–337, 1994.
Constructing
1988.
formal

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *J. Automated Reasoning*, **18**:237–246, 1997.

[MMM97] A. Mili, R. Mili, and R. Mittermeir. "Storing and Retrieving Software Components: A Refinement-Based System". *IEEE Trans. Software Engineering*, **SE-23**(7):445–460, 1997.

[MW95] A. Moorman Zaremski and J. M. Wing. "Specification Matching of Software Components". In *Proc. 3rd ACM SIGSOFT Symp. Foundations of Software Engineering*, pp. 6–17, Washington, Oct. 1995. ACM Press.
Comp.

[PBA95] J. Penix, P. Baraona, and P. Alexander. "Classification and Retrieval of Reusable Components Using Semantic Features". In *Proc. 10th Knowledge-Based Software Engineering Conf.*, pp. 131–138, Boston, Nov. 1995. IEEE Comp. Soc. Press.
1991.

[RS98] W. Reif and G. Schellhorn. "Theorem Proving in Large Theories". In Bibel and Schmitt [BS98]. To Appear.
Retrieval
Artifical

[WGR96] C. Weidenbach, B. Gaede and G. Rock. "Spass and Flotter version 0.42". In Proc. CADE-13, LNAI 1104, pp. 57-60, Springer, 1996.