SETHEO Goes Software Engineering: Application of ATP to Software Reuse*

Bernd Fischer[1] and Johann M. Ph. Schumann[2]

[1] Abt. Softwaretechnologie, TU Braunschweig, D-38092 Braunschweig
fisch@ips.cs.tu-bs.de
[2] Institut für Informatik, TU München, D-80290 München
schumann@informatik.tu-muenchen.de

Reuse of approved software components has been identified as one of the key factors for successful software engineering projects. Although the reuse process also covers many non-technical aspects [9] we will restrict ourselves to the retrieval of software components (SCR) based on their formal specifications. Our system NORA/HAMMR[3] is based on a library of software components with associated specifications of their pre- and postconditions written in VDM-SL [2]. A query consists of pre- and postconditions ($pre_q, post_q$) and the signature of the desired component. "Plug-in-compatibility" of a library component c is guaranteed, if $(pre_q \Rightarrow pre_c) \wedge (post_c \Rightarrow post_q)$ can be shown.

This "retrieval-by-proof" or *deduction-based* approach to SCR has been proposed before (e.g., [7, 5]) but without convincing success. These earlier failures result from the strong application requirements, like critical ("sub-minute") response times and full automatic processing: the proof tasks must be generated and processed completely automatically as we cannot expect the end-user to cope with ATP details.

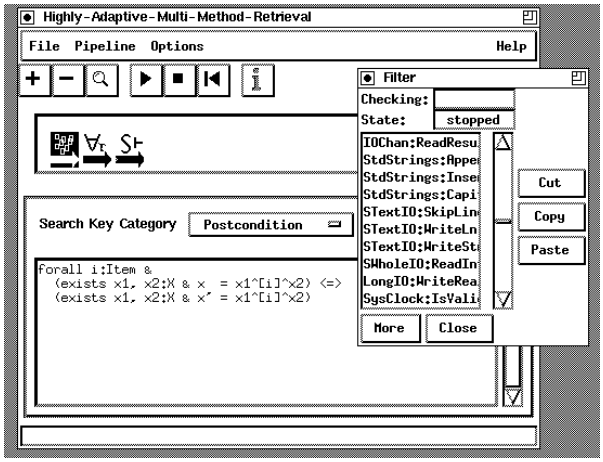
System Architecture. The naïve approach to deduction-based SCR would be to generate a separate proof task for each candidate. But then the number of tasks per query would be prohibitively large. Even worse, most of them would be non-theorems (i.e., not valid) because in general, there are many more non-matching than matching candidates. We thus implemented NORA/HAMMR as a *filter pipeline* through which the candidates are fed.

NORA/HAMMR's graphical user interface (see figure on the next page) reflects the idea of successive filtering. The pipeline may easily be customized through an icon pad; additional inspectors display intermediate results and grant easy access to the components. The objective of the GUI is to hide all evidence of ATP usage. Hence, the knowledge necessary to use NORA/HAMMR as a *tool* is restricted to VDM-SL and the target language.[4]

* This work is supported by the DFG within the Schwerpunkt "Deduktion", the Sonderforschungsbereich 342, Subproject A5 (Parallelization of Inference Systems), and grant Schu-908/5-1.

[3] NORA is no real acronym, HAMMR is a highly adaptable multi-method retrieval tool.

[4] But we assume the existence of a reuse administrator who knows the applied deduc-



The pipeline typically starts with *signature matching filters*. They check whether candidate and query have "compatible" calling conventions (see [3] for a detailed discussion).

Then, *rejection filters* try to eliminate non-matches as fast as possible. We currently use model generation techniques to check the validity of the tasks in suitable finite models. However, both *precision* ("do we get the right components?") and *recall* ("do we get all matching components?") may decrease in an acceptable way because this approach is neither sound nor complete.

Finally, *confirmation filters* check the validity of the remaining proof tasks. Here, we apply SETHEO, a high-performance automated theorem prover based on the Model Elimination Calculus. SETHEO accepts unsorted first-order logic (without equality) in clausal normal form and tries to refute the formula by constructing a tableau (cf. e.g. [4, 6]). In the remainder of this paper we will describe how SETHEO handles these proof tasks without having an intolerable decline of recall.

Characteristics of Proof Tasks. Let us consider the following VDM-SL specifications:[5]

$rotate(l : List)\ l' : List$	$shuffle(x : X)\ x' : X$
pre true	pre true
post $(l = [] \Rightarrow l' = [])\wedge$	post $\forall i : Item \cdot (\exists x_1, x_2 : X \cdot x = x_1\hat{}[i]\hat{}x_2 \Leftrightarrow$
$\quad (l \neq [] \Rightarrow l' = (\mathsf{tl}\ l)\hat{}[\mathsf{hd}\ l])$	$\quad \exists x_1, x_2 : X \cdot x' = x_1\hat{}[i]\hat{}x_2)$

Several steps have to be performed to convert these specifications (using *rotate* as candidate c and *shuffle* as query q) into a sorted first-order proof task. First, the formal parameters must be identified, in this case $l = x$ and $l' = x'$.[6]

---

tion methods and "tunes" (e.g., by giving ATP settings and lemmata) each component before it is entered into the library.

[5] Here, $\hat{}$ means the concatenation of lists, [] the empty list, $[i]$ a singleton list with item i, and hd and tl the functions head and tail, respectively.

[6] This identification is, however, not always a simple renaming substitution as VDM-SL allows pattern matching and complex data types.

Then, the main body of the proof task in the form $(pre_q \Rightarrow pre_c) \wedge (post_c \Rightarrow post_q)$ is constructed. Here, we have to make some assumptions on the definedness of specifications and the usage of partial functions, since VDM-SL's semantics is based on the three-valued logics of partial functions [2]. Finally, after several additional syntactical preprocessing steps, we arrive at the proof task

$$\begin{array}{l}\forall l, l', x, x' : \mathit{List} \cdot (l = x \wedge l' = x' \wedge \mathsf{true} \Rightarrow \mathsf{true}) \\ \quad \wedge (l = x \wedge l' = x' \wedge (l = [] \Rightarrow l' = []) \wedge (l \neq [] \Rightarrow (l \neq [] \Rightarrow l' = (\mathsf{tl}\ l)^\frown[\mathsf{hd}\ l])) \\ \qquad \Rightarrow (\forall i : \mathit{Item} \cdot (\exists x_1, x_2 : X \cdot x = x_1{}^\frown[i]{}^\frown x_2 \Leftrightarrow \exists x_1, x_2 : X \cdot x' = x_1{}^\frown[i]{}^\frown x_2)))\end{array}$$

Using SETHEO. For the final stage of the filter chain, the prover SETHEO is used to process the remaining proof tasks. A number of important items had to be considered for this application, e.g., handling of equality, incorporation of sorts, selection of axioms and case splitting. A proof task consists almost entirely of equations. Most equations are simple and just equate formal parameters, but the remaining equalities relate the functions' results to their respective arguments and thus specify the computational effect of the component. Thus, an efficient handling of equality is of particular importance. We currently provide two variants: the naïve approach by automatically adding the appropriate axioms of equality, and the handling of equality as it is done in E-SETHEO [6] (a transformation similar to Brand's STE-modification).

The hierarchy of sorts contained in the given proof tasks can be expressed as a many-sorted problem with constant sorts. This allows the given sort information to be compiled efficiently into the terms of the unsorted clauses. In our prototype, this is accomplished by the tool ProSpec (developed within PROTEIN [1]).

Due to their recursive structure, many proof tasks require induction to be solved properly. Our severe restriction on run-times, however, does not allow to use an inductive theorem prover. We thus approximate induction by *splitting* a proof task into several cases. For example, for a component and query with signature $c(l : \mathit{List})$, we generate cases for $l = []$, $l = [i]$, and $l = [i]^\frown l_0$ for item i and list l_0. This approximation cannot handle all inductive problems but since the base cases in general induce simple proof problems, this approach also allows us to implement additional powerful and fast filters to eliminate proof tasks.

Finally, when SETHEO is started with the pre-processed proof task, it has only a few seconds of run-time (wall clock) to find a proof. Therefore, we use *parallel competition*: on each available processor, SETHEO is started with a different parameter setting. The one which first returns with a proof "wins" and aborts the others (cf. also SiCoTHEO [8]).

Experiments. First experiments with NORA/HAMMR and SETHEO have been carried out with more than 350 proof tasks over components about lists. All experiments have been carried out on a SUN Ultra-SPARC 2 (competition with 4 different sets of parameters) with a CPU-time limit of 20 seconds. First, we tried to retrieve components with identical specifications. Here, no axioms about the theory are necessary. Except for three error cases (due to excessive size of the input formula), all 55 provable tasks could actually be solved by SETHEO

($T_{\text{mean}} = 0.43$ seconds). Additional experiments tried to retrieve matching but non-identical components (38 cases in our test-library). With the help of case-splitting, 21 of these tasks could be solved within our time limit of 20 seconds, yielding a total recall of 63.7%. Here, however, the selection of axioms turned out to be crucial. Therefore, our prototype currently just selects those axioms which directly share function symbols with the proof task.[7]

Conclusions. The results of our experiments with SETHEO are very encouraging. Nevertheless, many improvements on this approach have to be made before this tool can really be used in industry. Due to the hard time-constraints ("results while-u-wait"), the reduction of proof-tasks, both in complexity and number is of central importance. A powerful chain of filters (ranging from signature matching to dis-proving techniques) must ensure that only a few proof tasks remain to be processed by the automated theorem prover. For these, a set of axioms as small as possible and valuable lemmas must be selected. Furthermore, work will be done to improve the approximation of induction.

This application of automated theorem proving technique carries the unique feature that soundness and completeness are not absolutely vital — unsound and incomplete methods only reduce the precision and recall of the retrieval tool. This allows interesting and promising techniques of approximating proofs (e.g., by filter chains or iteration) to be explored which will help to lead to an industrial application of SETHEO.

[1] P. Baumgartner and U. Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In *Proc. 13th Conf. Automated Deduction*. Springer, 1996.

[2] J. C. Bicarregui, J. S. Fitzgerald, P. A. Lindsay, R. Moore, and B. Ritchie. *Proof in VDM: A Practitioner's Guide*. Springer, 1993.

[3] B. Fischer. *A systematic approach to type-based software component retrieval*. PhD thesis, TU Braunschweig (in preparation).

[4] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *J. Automated Reasoning*, 8(2):183–212, 1992.

[5] P. Manhart and S. Meggendorfer. A knowledge and deduction based software retrieval tool. In *Proc. 4th Intl. Symp. Artificial Intelligence*, pp. 29–36, 1991.

[6] M. Moser et.al. The Model Elimination Provers SETHEO and E-SETHEO. Special issue of *J. Automated Reasoning*, to appear 1997.

[7] E. J. Rollins and J. M. Wing. Specifications as search keys for software libraries. *Proc. 8th Intl. Conf. Symp. Logic Programming*, pp. 173–187. MIT Press, 1991.

[8] J. Schumann. SiCoTHEO: Simple Competitive parallel Theorem Provers (System Abstract). In *Proc. 13th Conf. Automated Deduction*. Springer, 1996.

[9] M. Zand and M. Samadzadeh. Software reuse: Current status and trends. *J. Systems Software*, 30(3):167–170, 1995.

This article was processed using the LaTeX macro package with LLNCS style

[7] Although this approach is not complete, we rather aim at finding proofs for many (obvious) proof tasks within the given short run-time. With too many axioms, SETHEO is not able to find any proofs at all within our short time-limits.