JOHANN SCHUMANN

AUTOMATED THEOREM PROVING IN HIGH-QUALITY SOFTWARE DESIGN

1. INTRODUCTION

The amount and complexity of software developed during the last few years has increased tremendously. In particular, programs are being used more and more in embedded systems (from car-brakes to plant-control). Many of these applications are safety-relevant, i.e. a malfunction of hardware or software can cause severe damage or loss. Tremendous risks are typically present in the area of aviation, (nuclear) power plants or (chemical) plant control (Neumann, 1995). Here, even small problems can lead to thousands of casualties and huge financial losses. Large financial risks also exist when computer systems are used in the area of telecommunication (telephone, electronic commerce) or space exploration. Computer applications in this area are not only subject to safety considerations, but also security issues are important.

All these systems must be designed and developed to guarantee *high quality* with respect to safety and security. Even in an industrial setting which is (or at least should be) aware of the high requirements in Software Engineering, many incidents occur. For example, the Warshaw Airbus crash (Neumann, 1995), pg. 46, was caused by an incomplete requirements specification. Uncontrolled reuse of an Ariane 4 software module was the reason for the Ariane 5 disaster (Lions et al., 1996). Some recent incidents in the telecommunication area, like illegal "cloning" of smart-cards of D2-GSM handies (Spiegel, 1998), or the extraction of (secret) passwords from German T-online users (c't, 1998) show that also in this area serious flaws can happen.

Due to the inherent complexity of computer systems, most authors claim that only a *rigorous* application of *formal methods* in all stages of the software life cycle can ensure high quality of the software and lead to real safe and secure systems. In this paper, we will have a look, in how far automated theorem proving can contribute to a more widespread application of formal methods and their tools, and what automated theorem provers (ATPs) must provide in order to be useful. We will justify our observations with results of case studies, most of which have been carried out with the theorem prover SETHEO (Letz et al., 1992; Goller et al., 1994).

2. FORMAL METHODS AND DEDUCTION

Formal methods in general refer to the use of techniques from logic and discrete mathematics in specification, design and construction of computer systems and software (Kelly, 1997). Formal methods are based on logic and require the explicit and concise notation of all assumptions. Reasoning is performed by a series of inference steps of the underlying logic (formal proof).

Formal methods can be applied during various stages of the software life cycle and on different levels of "formality". (Kelly, 1997), pg. 7 distinguishes between three levels of formalization: on the lowest level, mathematical concepts and notations are used to express the requirements and assumptions. However, the analysis (if there is any) is only performed in an informal way. On the second level, formal specification languages are located. Also based on mathematical concepts, the underlying (denotational or operational) semantics of the specification language allows to perform formal reasoning. On this level, we find the classical specification languages like Z, VDM, and others, some with computer support (like syntax-controlled editors, type-checkers, or simulators). Finally, the third level concerns formal specification languages with a comprehensive environment including (automated) theorem provers and proof checkers.

Of course, the effort spent on formal methods substantially increases as the level of formalization rises. However, for the design of High-Quality Software, a considerable level of formal reasoning (with computer support) is necessary. Only then, even intricate errors can be detected, and safety and security properties can be guaranteed. Such a level usually requires proving lots of theorems on a very formal, detailed level. Doing this by hand is not only a very time-consuming, but also error prone task. Hence, computer support for (or, ideally automatic processing of) the proof obligations is necessary. Practical application of formal methods in industry (see e.g. (Weber-Wulff, 1993)), however, poses additional requirements on methods and tools: they must be *usable* and *user friendly*. This implies that a tool should

- support the entire software life cycle,
- fit smoothly into existing software development procedures,
- exhibit a fast learning curve,
- hide non-problem-specific details (e.g., existence of a prover), and
- be suited for real applications.

The last issue means that, for example, a tool must rather be able to handle trivial, but lengthy code (e.g., legacy code, macro code) than dealing with complex recursive algorithms (just think at the famous "quick-sort" example). Today's tools supporting formal methods are using *interactive theorem provers, model checkers*, and to a less extend, *automated theorem provers*.

2.1. Interactive Theorem Provers

Traditionally, interactive theorem provers like ACL2 (Kaufmann and Moore, 1996), EVES (Craigen and Saaltink, 1996), HOL (Gordon, 1987), Isabelle (Paulson, 1994), KIV (Reif, 1992), NqThm (Boyer and Moore, 1988), and PVS (Crow et al., 1995) — just to name a few — are being used to tackle proof tasks arising in many applications. These systems have a highly expressive input language. A higher order logic or customized logic can be defined formally and used within this framework. Formal definitions of theories can be used for the semi-automatic generation of induction schemes and simplifiers. Interactively activated *tactics* process the goals of the current theorem to be proven. Most systems furthermore contain an interactive (mostly emacs-based) user interface which allows to work on the open goals and to control the data base of theorems already proven.

Interactive theorem provers can be customized for specific applications and domains. To this end, the prover is augmented with definitions of specific logics and libraries of domain-specific tactics. Nevertheless, the proof of a theorem in general requires many interactions. Proof times of several months (e.g., (Schellhorn and Ahrendt, 1998; Havelund and Shankar, 1996)) are not an exception. Furthermore, the user must have a detailed know-how of the custom logic and the prover itself. For example, many proofs in (Paulson, 1997b) "[. . .] require deep knowledge of Isabelle". Despite their power, interactive theorem provers are only of limited usability in an industrial environment, because of their long learning curve and their relatively little degree of automatic processing.

2.2. Model Checking

On the other hand, Model Checkers for propositional (temporal) logic are more and more used in important applications. Originating from the area of hardware design and verification, these automatic tools provide an efficient means to tackle (large) proof tasks which have a finite state space (e.g., finite automata). Prominent systems are e.g., SMV (Burch et al., 1992), SPIN (Holzmann, 1991), Step, murphi (Dill, 1996), or μcke. Logics, specifically suited for the description of finite automata and their properties (e.g., CTL (Burch et al., 1990)) and convenient input languages facilitate the use of Model Checkers. Their ability to generate *counter-examples* when a conjecture cannot be proven provides valuable feed-back for the user.

Recently, Model Checkers have been extended (see e.g. (Burkart, 1997) or Mona (Klarlund and Møller, 1998)) to handle infinite domains which have a finite model property (i.e., if a model exists it has a finite size). Nevertheless, Model checkers usually cannot be applied in applications where recursive functions and data structures are used. Furthermore, most systems are not able to produce a formal proof (as a sequence of inference steps) which can be checked (or proof-read) externally (but see e.g., the Concurrency Workbench (Moller, 1992; Cleaveland et al., 1993)). Rather, the user has to rely on the correctness of the implementation[1]. The most severe reduction for the practical applicability of Model Checkers is the limit of the size of the state space they can handle. Despite numerous approaches (e.g., (Clarke et al., 1994)), proof tasks must be broken down or abstracted carefully in order to avoid state space explosion.

2.3. Automated Theorem Provers

Automated theorem provers (ATPs) for first order predicate logic (e.g. OTTER (McCune, 1994a), Gandalf (Tammet, 1997), METEOR (Astrachan and Loveland, 1991), SETHEO (Letz et al., 1992; Letz et al., 1994; Moser et al., 1997), SNARK (Stickel et al., 1994), SPASS (Weidenbach et al., 1996) ...[2]) can handle full first order logic. Nevertheless, they are only used very rarely in applications in the area of Software Engineering. Although, due to intensive research (e.g., the German "Schwerpunkt Deduktion" (Bibel and Schmitt, 1998)), these systems have gained tremendously in power, one is tended to ask: "Why are they not used?" and "Is there really a gap between Higher-Order logic interactive theorem proving[3] and decision procedures?"

Currently, most Automated Theorem Provers (ATPs) are like racing cars: although very fast and powerful, they cannot be used for everyday traffic, because essential things (like head-lights) are missing. The classical architecture of an ATP (i.e., a highly efficient and tuned search algorithm) will and must be extended into several directions in order to be useful for real applications.

For the rest of this paper, we are concerned with the topic how, and in which way automated provers are to be extended in order to be applicable in the area of High Quality Software Design. We can identify *direct application* (i.e., proof obligations are already suited for direct processing by an ATP), *integration* of ATPs into interactive theorem provers, and the *adaptation* of automated provers towards practical applicability.

3. DIRECT APPLICATIONS

Racing tracks are specifically suited for racing cars. In our application area, domains can be identified which are suited for direct processing with an ATP. Obviously, the formal method should be close to First Order Logic (FOL), or a logic which can be translated effectively into FOL. Furthermore, the proof obligations must be of a complexity (size of the formula and size of the induced search space) which can be handled within current-technology theorem provers and the proof tasks must be provable without application of induction[4]. If these requirements are not met, it might be better not to use a general purpose ATP, but some special purpose algorithm. E.g., in PLANWARE (Burstein and Smith, 1996) the designers had enough information about how to find a solution that they "were able to throw out the theorem prover" (Smith, 1998). In this paper, we will have a look at three specific systems which directly apply automated theorem provers: PIL/SETHEO, NORA/HAMMR, and AMPHION.

3.1. PIL/SETHEO

PIL/SETHEO is a prototypical tool for the automatic verification of authentication protocols (Schumann, 1997; Dahn and Schumann, 1998; Schumann, 1999b). Authentication protocols are used in most distributed applications (e.g., internet, electronic commerce, wireless telephone networks) to identify the communication partners and to establish a secure communication, e.g., by exchanging encryption keys. Due to the importance of these protocols, their verification is vital. Several formal approaches to guarantee security properties (c.f. (Meadows, 1994; Geiger, 1995) for an overview) have been developed in the past: using modal logics of belief (e.g., (Burrows et al., 1989; Burrows et al., 1990; Gong et al., 1990; Syverson and van Oorschot, 1994; Abadi and Tuttle, 1991; Kessler and Wedel, 1994; Kindred and Wing, 1996)), Model Checking (Kindred and Wing, 1996), and approaches based on communicating sequential processes (Paulson, 1997a; Paulson, 1997b; Lowe, 1996).

PIL/SETHEO can analyze authentication protocols using the modal BAN-logic (Burrows et al., 1989) or the AUTLOG-logic (Kessler and Wedel, 1994). These logics are often employed in early stages of protocol development, because they are able to handle freshness properties (an important class of security properties) and produce short and informative proofs. Given the specification of a protocol and additional assumptions, PIL/SETHEO transforms the formulas into first order logic and starts the prover SETHEO. Proofs found by SETHEO are then automatically translated into a problem-oriented, human-readable form. An example for input and output for a simple protocol (a slight modification of the RPC-handshake (Satyanarayanan, 1987)) is shown in Figure 1. Proofs are in general found within a few seconds of run-time. Rather than going into details (which can be found in (Schumann, 1997; Dahn and Schumann, 1998)), Figure 1 illustrates an extremely important feature of PIL/SETHEO (as of any successful application): it hides any evidence of the automated theorem prover and first order logic. Both, input and output are in problem-oriented form (here, a modal belief logic), and the user does not need to have any knowledge about the prover's details.

3.2. NORA/HAMMR

Reuse of approved software components is an important method for ensuring high quality in software systems. NORA/HAMMR (Schumann and Fischer, 1997; Fischer et al., 1998) is a tool for the deduction-based retrieval of components from a library of reusable components. Using a contract-based approach, the library modules are identified by a VDM/SL specification of their pre- and post-conditions. In order to retrieve components, a query specification (also in VDM/SL) is given. Then, the library is automatically searched for matching modules. Matching components usually have a weaker pre-condition than the query, and a stronger post-condition (for details on different ways of retrieval see (Fischer, 1999)).

The tool NORA/HAMMR is designed for optimal usability. A graphical user interface (see Figure 2) allows to enter the query, browse selected components from

```
Objects:
 principal A,B;
 sharedkey K_a_b, Kp_a_b;
 statement N_a, N_b;
Assumptions:
 A believes sharedkey K_a_b;
 B believes sharedkey K_a_b;
 A believes B controls
      sharedkey K_a_b;
 B believes sharedkey Kp_a_b;
 A believes fresh N_a;
 B believes fresh N_b;
Idealized Protocol:
 message 1: A -> B
      {N_a}(K_a_b);
 message 2: A <- B
      {f(N_a),N_b}(K_a_b);
 message 3: A -> B
      {N_b}(K_a_b);
 message 4: A <- B
      {sharedkey Kp_a_b}
         (K_a_b);
Conjectures: after message 4:
 B believes A believes N_b;
```

Theorem 1. $conjecture.$

Proof. We show directly that

$$conjecture. \tag{1}$$

Because of *Message-Meaning*, *Assumption*$_2$, and by *Message*$_3$

$$\vdash B \mid\equiv A \mid\sim N_B. \tag{2}$$

Because of *Theorem*

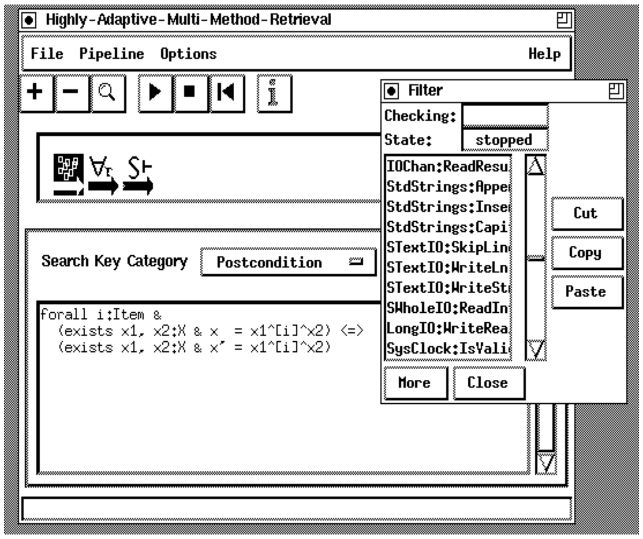
$$conjecture \Leftarrow \vdash B \mid\equiv A \mid\equiv N_B. \tag{3}$$

Because of *Nonce-Verification*: $A \mid\equiv B \mid\equiv C \Leftarrow A \mid\equiv B \mid\sim C \wedge A \mid\equiv \# C$. Hence by (2) and by *Assumption*$_6$ $\neg\, conjecture$. Hence by (3) $conjecture$. Thus we have completed the proof of (1). q.e.d.

Figure 1 Example input and output of PIL/SETHEO

the library, and control the search process which is structured as a *pipeline* of filters of increasing deductive power. The filters are capable of efficiently discarding non-matching components. Starting with signature matchers and simplification filters, most of the interesting components can be preselected. A subsequent model checking filter tries to identify non-matching components. Finally, a first-order theorem prover processes the matching relation. If a proof can be found, the library component can be safely reused. Hence, only a tiny fraction of proof tasks is processed by the automated prover. This ensures *short answer-times*, one of the most important requirements for this kind of applications.

The filter pipeline can be configured by the user, but the control of the provers is kept invisible from the user. Therefore, the tool only requires knowledge about the (problem-oriented) specification language VDM/SL. NORA/HAMMR (see Figure 2 for a screen-shot) has been evaluated extensively on a large library on functions about lists (Fischer et al., 1998). The automated provers Protein (Baumgartner and Furbach, 1994), SETHEO and SPASS (Weidenbach et al., 1996) have been used for the experiments. With a run-time limit of 60 seconds, a recall (percentage of retrieved matching components) of more than 71% (SPASS) could be obtained. If all provers are


Highly-Adaptive-Multi-Method-Retrieval
File Pipeline Options Help
Filter
Cut
Copy
Paste
More
Close

Checking:
State: stopped

```
IOChan:ReadResu
StdStrings:Appe
StdStrings:Inse
StdStrings:Capi
STextIO:SkipLin
STextIO:WriteLn
STextIO:WriteSt
SWholeIO:ReadIn
LongIO:WriteRea
SysClock:IsVali
```

Search Key Category Postcondition

```
forall i:Item &
  (exists x1, x2:X & x  = x1^[i]^x2) <=>
  (exists x1, x2:X & x' = x1^[i]^x2)
```

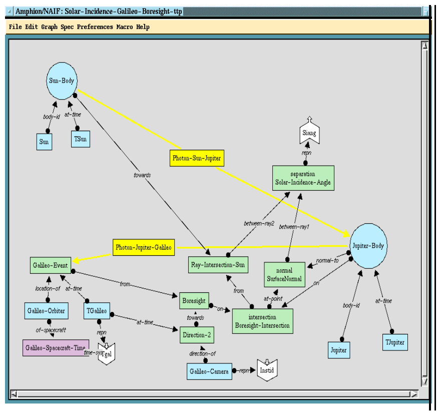
Figure 2 Graphical User Interface of NORA/HAMMR

running in a parallel competitive mode (see Section 5.2), the recall could be increased to 80% which is acceptable for practical purposes.

3.3. Amphion

The main application area of AMPHION (Lowry et al., 1994) is the automatic synthesis of astrodynamic (FORTRAN-) programs out of a given subroutine library (NAIF). Specifications are entered in a graphical way as shown in Figure 3 (taken from (Lowry et al., 1998)). All bodies (here Jupiter, Sun, space-craft) and their relationship as well as the desired function (here to calculate the boresight angle between the Space-craft Galileo and the Sun) are entered using graphical elements. From this, an internal formal specification of the problem is automatically generated and processed by the automated prover SNARK (Stickel et al., 1994). Its result corresponds to the sequence of library calls necessary to calculate the desired function. Finally, a post-processor converts this data structure into the desired FORTRAN program (Figure 3).

This tool is widely used within NASA and has been extended to handle several other domains. Here again, hiding the prover and its logic is important. Only then, this tool can be used by non-specialists in the area of theorem proving. Further features of this kind of application come immediately into mind: proofs as a result are important, the domain is rather restricted (only a linear sequence of subroutine calls) but nevertheless important. However, more complex application domains probably would require at least some kind of user interactions and/or specific adaptations and extensions to bare-



```
   SUBROUTINE SOLAR0 (TGAL,INSTID,SIANG)
C  ...
C       Input variables
   CHARACTER*(*) TGAL
   INTEGER INSTID
C       Output variables
   DOUBLE PRECISION SIANG
C  ...
   CALL SCS2E ( GALILE, TGAL, ETGALI )
   CALL BODVAR ( JUPITE, 'RADII', DMY1, RADJUP )
   CALL SPKSSB ( GALILE, ETGALI, 'J2000', PVGALI )
   CALL SCE2T ( INSTID, ETGALI, TKINST )
   TJUPIT = SENT ( JUPITE, GALILE, ETGALI )
   CALL BODMAT ( JUPITE, TJUPIT, MJUPIT )
   CALL ST2POS ( PVGALI, PPVGAL )
   CALL SPKSSB ( JUPITE, TJUPIT, 'J2000', PVJUPI )
C  ...
   CALL SURFNM ( RADJUP(1), RADJUP(2), RADJUP(3), P, PP)
   CALL MTXV ( MJUPIT, P, XP )
   CALL MTXV ( MJUPIT, PP, XPP )
   CALL VADD ( PPVJUP, XP, V0 )
   CALL VSUB ( PPVSUN, V0, DV0PPV )
   SIANG = VSEP ( XPP, DV0PPV )
   RETURN
   END
```

Figure 3 Example input and output of AMPHION

bone automated theorem provers (e.g., inclusion of decision procedures (Lowry and van Baalen, 1995)). Both directions will be discussed in the following.

4. INTEGRATION INTO INTERACTIVE PROVERS

The goal of integrating automated theorem provers into interactive provers is to relieve the user from tedious, error-prone and low-level work. Only major decisions — the "central proof ideas" — will have to be provided by the human user, whereas trivial tasks are performed automatically. Such a system architecture can profit much from the reasoning power of automated theorem provers. For typical applications, e.g., in verification, more than about 90% of the Higher-Order logic constructs[5] can easily be transformed into one or more first order proof tasks. According to (Reif, 1998), ideally about 25–30% of these tasks can be handled automatically. The others must be further broken down by the user. We will have a look at two prominent systems which combine interactive theorem provers with high-performance automated theorem proving: ILF and KIV.

4.1. ILF

ILF (Dahn et al., 1994) is an interactive proof environment ("Proof-Pad") which allows the interactive construction of complex proofs by using tactics. The system has been successfully applied to problems from mathematics (Mizar (Dahn and Wernhard, 1997)), hardware verification (verification of a microprocessor (Wolf and Kmoch, 1997)) and the verification of communication protocols (Dahn and Schumann, 1998).

This work showed its usability in the area of High-Quality software and hardware design.

For the verification of software protocols, an interface to the specification language Z has been defined and implemented (Dahn and Schumann, 1998). Z specifications are directly translated into ILF's sorted first order predicate logic. With the help of a graphical user interface and a tactics-based language, the proof tasks can be broken down into individual subgoals. Each subgoal is first tried by one of the connected automated provers (OTTER (McCune, 1994a), SETHEO (Letz et al., 1992; Goller et al., 1994), and Discount (Denzinger, 1995; Denzinger and Pitz, 1992)[6]). The time-limit for the automatic tries — which are performed in parallel (see Section 5.2) — is set to several seconds. If during this time no proof could be found, the user has to further break down the proof obligation.

A unique feature of ILF is its capability to present the user a problem-oriented, human readable proof. Such a proof consists of a "patchwork" of small proofs, found by the automated provers and by the interactive application of tactics. By using a common (natural deduction-style) calculus, the Block calculus (Dahn and Wolf, 1994), all proofs can be represented in a uniform style. This proof is then automatically post-processed and typeset to obtain a LaTeX-document (see also Figure 1, right-hand side for an example).

4.2. KIV and Automated Theorem Provers

KIV (Reif et al., 1997) is an interactive verifier, specifically suited for the verification of High-Quality software. Many industrial applications of KIV, e.g., the verification of an Airbag controller (Reif, 1998) demonstrates the practical usability of this system. Based on dynamic logic, reasoning can be performed manually and by tactics. As with any interactive system, the time to find a proof can be considerably. For example, for the verification of certain refinement steps for a PROLOG Abstract Machine implementation (Börger and Rosenzweig, 1995), proof times of up to two months have been reported in (Schellhorn and Ahrendt, 1998)[7].

Hence, one aim of KIV's developers is to use automated theorem provers to process simple proof tasks without user interaction. To this end, the prover $_3T^AP$ (Hähnle et al., 1992; Hähnle, 1993; Beckert and Hähnle, 1992) was integrated into KIV, and experiments on the expected performance of SETHEO, SPASS, Otter, and Protein (Baumgartner and Furbach, 1994) have been performed for selected domains (Schellhorn and Reif, 1998). One key-problem which was identified is the preprocessing of axioms: when working with the interactive system, one usually loads all theories (e.g., theory of natural numbers and arithmetic, lists, trees) which might be needed for the ongoing verification. When interactive steps are performed, the human user usually knows which of the axioms is to be applied. Most automated theorem provers, however, are overwhelmed by the sheer number of axioms (often several hundreds) which are included in the formula. Therefore, a powerful mechanism for the preselection of axioms is vital. A straight-forward method has been integrated into KIV (see Section 5.1.1) which performs very well.

5. ADAPTATION OF AUTOMATED PROVERS

Although automated theorem provers for first order logic have become very powerful, they need to be adapted in order to be useful for the intended application. This is due to the fact that most ATPs have been designed and evaluated with the aim to solve small, but hard problems as they typically occur in mathematics. The large collection of problems in the TPTP benchmark library (Sutcliffe et al., 1994) reflects this fact: most problems are specifically formalized and prepared for automated deduction. Proof tasks from applications in the area of high quality software design usually poses quite different requirements for the automated theorem prover (cf. (Schumann, 1999a)). Extensions needed to meet these requirements include handling of non-first order logics (e.g., inductive problems, modal logics), efficient equality and theory treatment, handling of non-theorems (i.e., giving useful feed-back if a conjecture cannot be proven), post-processing of proofs, etc. Due to space restrictions we will focus on only two topics: preprocessing of the formulas and parallel execution in order to reduce the prover's answer time. Although all extensions mentioned above are important, experience with many case studies (cf. (Schumann, 1999a)) revealed that these issues are central for a successful application.

5.1. Preprocessing

In most applications, proof obligations are generated automatically by the application system. Because this usually involves a transformation between different logics, the generated formulas can contain many redundancies and lots of axioms which will not be needed for finding a proof. Automated theorem provers, however, are extremely sensitive with respect to the number of (unnecessary) clauses added to the formula. Therefore, it is important to perform a powerful *preprocessing* of the formula with the goal of optimizing it without affecting its provability.

5.1.1. Preselection of Axioms

In general, the task of selecting the appropriate axioms which contribute to the proof is as hard and undecidable as the proof task itself. Therefore, approximations are necessary. For domains with a rich signature and a hierarchical structured theory, a straight-forward and powerful method has been developed in (Schellhorn and Reif, 1998): given a formula, one selects only such (sub-)theories which are on the branch from the theory, required by the theorem to the root of the hierarchy. For example, to prove a theorem about the length of a list using the append operator, only the axioms belonging to the sub-theories of lists and natural numbers with addition have to be added. So, axioms, defining multiplication (which usually also belong to the theory of natural numbers) can be omitted. This mechanism is a part of the interactive system KIV (Section 4.2) and has been evaluated in (Schellhorn and Reif, 1998) with theorems, coming from the domain of graphs. With the full set of axioms (more than 500 axioms), SETHEO could show 18 example theorems ((Schellhorn and Reif, 1998), pg. 238). The axiom reduction yields less than 100 axioms which enabled SETHEO to solve 29 problems[8]. Similar methods have been integrated into ILF and NORA/HAMMR.

5.1.2. Simplification

Proof tasks which are being generated automatically, typically contain parts which are not useful for the current proof obligation. Examples are parts which are obviously not usable ("pure" parts or tautologies, e.g., $\mathcal{F} \wedge$ TRUE). Such sub-formulas can be removed without affecting the logical properties of the formula, thus reducing the theorem prover's search space considerably.

Very old versions of SETHEO (Letz et al., 1988) contained such preprocessing modules. However, due to the general focus on problems which are already simplified and minimized, this aspect had been totally neglected by most ATP designers. Simplification should be performed as soon as possible (i.e., already on the level of the application) and on all subsequent levels (application logic, first order logic, clausal normal form). Typical ways of simplifying a formula which directly come into mind are:

- removal of obviously tautological parts (e.g., TRUE, FALSE, $p \vee \neg p$).
- optimization of the quantifies with the aim of producing Skolem functions with minimal length and small sets of clauses (cf. e.g. (Eder, 1985; Nonnengart et al., 1998)).
- simplification of the formula with respect to underlying theories. This means that e.g., $\mathsf{cons}(X, Y) = []$ can be simplified to FALSE.
- formulas of the form $\forall X : (X = a \rightarrow \mathcal{F}[X])$ and some arbitrary $\mathcal{F}$ can be replaced by: $\mathcal{F}[X \backslash a]$. This means that all unconditional equations are applied to the whole formula and then removed. This kind of simplification is particular helpful when processing induction cases like $\forall L : L = [] \rightarrow \mathcal{F}$.

However, much more sophisticated ways of simplification are possible and desirable. Some methods, using semantic information have been implemented in the system NORA/HAMMR (Fischer et al., 1998). Here, simplification is also used to decide (where possible) if the theorem can be valid or not. In particular in applications, where most of the proof obligations are non-theorems, this turns out to be extremely helpful. But this is by far not the end. Experience with interactive theorem provers and symbolic algebra systems (e.g., Mathematica (Wolfram, 1991)) reveals that simplifiers are *the* central and most intricate parts of such a system. Therefore, elaborate simplification of a proof task should not only be performed during the preprocessing phase, but also during the search for the proof.

5.2. Parallel Execution

Most applications of automated theorem provers share a common and important requirement: short answer time. In particular for systems like NORA/HAMMR, KIV, or AMPHION where the user is *waiting* for the answer ("results while-u-wait"), short response times are vital for the acceptance of such a tool. Furthermore, the automated prover hooked to the system should be able to handle reasonably complex proof obligations, and the behavior of the prover should be "smooth". This means that proof tasks which are somewhat similar to each other should also exhibit a similar behavior w.r.t. response times.

A paradigm which is able to support the above requirements is the exploitation of *parallelism*. As has been demonstrated in (Kurfeß, 1990), automated theorem proving exhibits an enormous potential of parallelism. With the widespread availability of modern architectures like coupled multi-processor systems and large (often idle) networks of powerful workstations, parallel processing of proof tasks has gained practical importance.

Of the many approaches of parallel theorem proving which have been explored (see e.g. (Suttner and Schumann, 1993; Schumann et al., 1998) for an overview), the models of *competition* and (static) *partitioning* seem to be the most appropriate ones for applications. In a competitive model, all processes must solve the same (entire) proof task, but can use different parameters (p-SETHEO (Wolf, 1998)) or parameter ranges (SiCoTHEO (Schumann, 1995)). The first process which finds a solution, wins and aborts the other processes. If the search parameters for each process exhibit a behavior which is sufficiently different from the others, good speed up values can be obtained while reducing the answer times. As an additional advantage, the model does not rely on high communication bandwidth and low latency, because interprocess communication is limited to start-up and shut-down of the system.

On the contrary, a static partitioning approach (e.g., SPTHEO (Suttner, 1995)) splits up the formula into many independent parts which are searched individually and in parallel. This model also produces good scalability and efficiency. In contrast to dynamic partitioning (like PARTHEO (Schumann and Letz, 1990)), static partitioning is easier to implement on arbitrary architectures and has substantially lower requirements on the communication means.

6. CONCLUSIONS

In this paper, we have investigated how first-order automated theorem provers can be applied to the development of High Quality Software. Formal methods which facilitate the design, development and verification need support by powerful, yet user-friendly tools. Here, automated theorem provers can be used to relieve the user from tedious, error-prone work on details of the proof obligations. We have identified two ways of applying ATP, namely direct applications and their integration into interactive provers and verifiers. However, automated provers without modifications can be applied only in very few cases, because they lack several important features. In this paper, we presented work on the following central issues: preprocessing of proof obligations and efficient control by exploitation of parallelism.

The enormous potential of automated provers can be used in practical applications only if important core requirements, identified in successful case studies are met:

- evidence of the automated prover must be hidden.
- automated provers must support handling of finite domains in an efficient way (e.g., by exploiting model generation techniques or by integrating decision procedures).
- ATPs must be able to handle *obvious* non-theorems appropriately, and must give feed-back in such cases (e.g., a counter-example).
- pragmatic issues must be obeyed more carefully, e.g., precisely defined input language, implementation restrictions (like reserved identifiers, length of symbols). In research-oriented environments, these issues are often overlooked and neglected. However, they are important prerequisites for real successful applications of automated theorem provers.

With carefully chosen application domains and theorem provers which meet the above requirements and are adapted accordingly, automated theorem provers are powerful enough to help to really bring forward industrial-applicable formal methods tools for the development of High-Quality Software.

ACKNOWLEDGMENTS

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) within the Habilitation-grant Schu908/5-1,Schu908/5-2,and SFB 342/A5. Partial funding for this paper was provided by NASA RTOP 519-50-22.

Johann Schumann
NASA Ames / Caelum Research Corp.
Moffett Field, Ca., USA

NOTES

1. This was also a point of critique when Model generators (Finder (Slaney, 1994) and Mace (McCune, 1994b)) were used in the area of finite quasi-groups.
2. See (Sutcliffe and Suttner (editors), 1997) for a broader overview.
3. Note, that some ITP's (e.g., PVS (Rajan et al., 1995)) already combine interactive theorem proving with decision procedures (e.g., model checking with μ-calculus and linear arithmetic solver).
4. Otherwise, special purpose inductive provers (e.g., Oyster/Clam (Bundy et al., 1990)) or interactive provers must be used to handle the proof task itself, or to generate first order subproblems ("base case, step case") out of the given proof task.

5. This figure results from estimates, given by several researchers to the author.

6. Whereas the first two provers can handle arbitrary formulas in first order logic with equality, Discount is restricted to problems which consist of equations only. In this domain, however, Discount is extremely powerful.

7. Although the topic of this case study might seem a little academic, its complexity and size resembles a typical demanding industrial application.

8. OTTER exhibits a similar behavior: it could solve 24 with all axioms and 31 with the reduced set of axioms ((Schellhorn and Reif, 1998), pg. 238).

REFERENCES

Abadi, M. and Tuttle, M. R. (1991). A Semantics for a Logic of Authentication. In *Proc. of the Tenth Annual ACM Symp. on Principles of Distributed Computing*, pages 201–216. ACM press.

Astrachan, O. and Loveland, D. (1991). METEORs: High Performance Theorem Provers using Model Elimination. In Boyer, R., editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe.* Kluwer Academic Publishers.

Baumgartner, P. and Furbach, U. (1994). PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In *Proc. 12th International Conference on Automated Deduction (CADE 12)*, volume 814 of LNAI, pages 769–773. Springer.

Beckert, B. and Hähnle, R. (1992). An Improved Method for Adding Equality to Free Variable Semantic Tableau. In Kapur, D., editor, *Proc. 11th International Conference on Automated Deduction (CADE 11)*, volume 607 of *LNAI*, pages 507 – 521. Springer.

Bibel, W. and Schmitt, P., editors (1998). *Automated Deduction: a Basis for Applications*, volume 8-10. Kluwer.

Börger, E. and Rosenzweig, D. (1995). The WAM — definition and compiler correctness. In *Logic Programming: Formal Methods and Practical Applications*, volume 11 of *Studies in Computer Science and Artificial Intelligence.* North-Holland.

Boyer, R. S. and Moore, J. S. (1988). *A Computational Logic Handbook.* Academic Press.

Bundy, A., van Harmelen, F., Horn, C., and Smaill, A. (1990). The Oyster-Clam System. In Stickel, M. E., editor, *Proc. 10th International Conference Automated Deduction (CADE 10)*, volume 449 of *Lecture Notes in Computer Science*, pages 647–648. Springer.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D., and Hwang, L. J. (1992). Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computing*, 98(2):142–170.

Burch, J. R., Clarke, E. M., McMillan, K. L., and Dill, D. L. (1990). Sequential Circuit Verification Using Symbolic Model Checking. In *Proc. 27th ACM/IEEE Design Autom. Conf.* IEEE Comp. Soc. Press.

Burkart, O. (1997). *Automatic Verification of Sequential Infinite-state Processes*, volume 1354 of *Lecture Notes in Computer Science.* Springer.

Burrows, M., Abadi, M., and Needham, R. (1989). A Logic of Authentication. In *ACM Operating Systems Review 23(5) / Proceedings of the Twelfth ACM Symposium on Operating Systems Principles.* ACM Press.

Burrows, M., Abadi, M., and Needham, R. (1990). A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.

Burstein, M. B. and Smith, D. (1996). ITAS: A Portable Interactive Transportation Scheduling Tool Using a Search Engine Generated from Formal Specifications. In *Proceedings of the 3rd International Conference on AI Planning Systems (AIPS-96)*, pages 35–44. AAAI Press.

Clarke, E. M., Grumberg, O., and Long, D. E. (1994). Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542.

Cleaveland, R., Parrow, J., and Steffen, B. (1993). The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72.

Craigen, D. and Saaltink, M. (1996). Using EVES to Analyze Authentication Protocols. Technical Report TR-96-5508-05, ORA Canada.

Crow, J., Owre, S., Rushby, J., Shankar, N., and Srivas, M. (1995). A Tutorial Introduction to PVS. In *WIFT'95 Workshop on Industrial strength formal specification techniques, Boca Raton, Fl, USA*.

c't (1998). T-online: Hacker knacken Zugangsdaten. *c't Computermagazin*, (7/98):62ff.

Dahn, B. I., Gehne, J., Honigmann, T., Walther, L., and Wolf, A. (1994). Integrating Logical Functions with ILF. Technical Report Preprint 94-10, Humboldt University Berlin, Department of Mathematics.

Dahn, B. I. and Wernhard, C. (1997). First Order Proof Problems Extracted from an Article in the MIZAR Mathematical Library. In *Proceedings of the 1st International Workshop on First-Order Theorem Proving (FTP)*, pages 58–62. RISC Linz, Austria.

Dahn, B. I. and Wolf, A. (1994). A Calculus Supporting Structured Proofs. *Journal for Information Processing and Cybernetics (EIK)*, (5–6).

Dahn, I. and Schumann, J. (1998). Using Automated Theorem Provers in Verification of Protocols. In Bibel, W. and Schmitt, P., editors, *Automated Deduction. A basis for applications*, chapter III.8, pages 195–224. Kluwer.

Denzinger, J. (1995). Knowledge-Based Distributed Search Using Teamwork. In *Proceedings ICMAS-95*, pages 81–88. AAAI-Press.

Denzinger, J. and Pitz, W. (1992). The DISCOUNT System. User Manual. SEKI Working Paper SWP-92-16, Universität Kaiserslautern.

Dill, D. L. (1996). The Murphi Verification System. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393. Springer.

Eder, E. (1985). An Implementation of a Theorem Prover based on the Connection Method. In Bibel, W. and Petkoff, B., editors, *AIMSA: Artificial Intelligence Methodology Systems Applications*, pages 121–128. North–Holland.

Fischer, B. (1999). *Deduction Based Software Component Retrieval.* PhD thesis, TU Braunschweig. (forthcoming).

Fischer, B., Schumann, J., and Snelting, G. (1998). Deduction based component retrieval. In Bibel, W. and Schmitt, P., editors, *Automated Deduction. A basis for applications*, chapter III.11, pages 265–292. Kluwer.

Geiger, J. (1995). Formale Methoden zur Verifikation kryptographischer Protokolle. Fortgeschrittenenpraktikum, Institut für Informatik, Technische Universität München. (in German).

Goller, C., Letz, R., Mayr, K., and Schumann, J. (1994). SETHEO V3.2: Recent Developments (System Abstract). In *Proc. 12th International Conference on Automated Deduction (CADE 12)*, volume 814 of LNAI, pages 778–782. Springer.

Gong, L., Needham, R., and Yahalom, R. (1990). Reasoning about Belief in Cryptographic Protocols. In *Proc. of IEEE Symposium on Security and Privacy, Oakland, Ca.,USA*, pages 234–248. IEEE.

Gordon, M. (1987). A proof generating system for higher-order logic. Technical Report 103, Univ. of Cambridge, Computer Laboratory.

Hähnle, R. (1993). *Automated Theorem Proving in Multiple–Valued Logics.* Oxford University Press.

Hähnle, R., Beckert, B., Gerberding, S., and Kernig, W. (1992). The Many–Valued Tableau–Based Theorem Prover $_3T^AP$. Technical report, IBM Germany Scientific Center Institute of Knowledge Based Systems.

Havelund, K. and Shankar, N. (1996). Experiments in Theorem Proving and Model Checking for Protocol Verification. In *FME '96, Oxford, UK*.

Holzmann, G. J. (1991). *Design and Validation of Computer Protocols.* Prentice Hall.

Kaufmann, M. and Moore, J. S. (1996). ACL2: An industrial strength version of NqThm. In *Compass'96: Eleventh Annual Conference on Computer Assurance.* National Institute of Standards and Technology.

Kelly, J. (1997). *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems. Volume II: A Practitioner's Guide.* NASA.

Kessler, V. and Wedel, G. (1994). AUTLOG — An Advanced Logic of Authentication. In *Proc. IEEE Computer Security Foundations Workshop IV*, pages 90–99. IEEE.

Kindred, D. and Wing, J. (1996). Fast, automatic checking of security protocols. In *2nd USENIX Workshop on Electronic Commerce*, pages 41–52.

Klarlund, N. and Møller, A. (1998). Mona version 1.2. User Manual. Brics technical report, BRICS, University of Aarhus, Denmark.

Kurfeß, F. (1990). *Parallelism in Logic — Its Potential for Performance and Program Development.* PhD thesis, Technische Universität München.

Letz, R., Mayr, K., and Goller, C. (1994). Controlled Integration of the Cut Rule into Connection Tableau Calculi. *Journal Automated Reasoning (JAR)*, (13):297–337.

Letz, R., Schumann, J., and Bayerl, S. (1988). SETHEO – A SEquential THEOremprover for first order logic. Technical report, ATP–Report, Technische Universität München.

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

Lions, J. L. et al. (1996). Ariane 5 flight 501 failure report.

Lowe, G. (1996). SPLICE-AS: A case study in using CSP to detect errors in security protocols. Technical report, Programming Research Group, Oxford.

Lowry, M. et al. (1998). The Amphion system. URL: http://ic-www.arc.nasa.gov/ic/projects/amphion.

Lowry, M., Philpot, A., Pressburger, T., and Underwood, I. (1994). Amphion: Automatic Programming for Scientific Subroutine Libraries. In *Proc. 8th Intl. Symp. on Methodology for Intelligent Systems, Charlotte, NC, USA*, pages 326–335.

Lowry, M. and van Baalen, J. (1995). META-AMPHION: Synthesis of efficient domain-specific program synthesis systems. In *Proceedings of the 10th Knowledge-Based Software Engineering Conference*, pages 2–10.

McCune, W. (1994a). OTTER 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Il, USA.

McCune, W. (1994b). A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, Argonne National Laboratory, Argonne, IL, USA.

Meadows, C. A. (1994). Formal verification of Cryptographic Protocols: A Survey. In *Proc. AsiaCrypt.*

Moller, F. (1992). *The Edinburgh Concurrency Workbench (Version 6.1)*. Department of Computer Science, University of Edinburgh.

Moser, M., Ibens, O., Letz, R., Steinbach, J., Goller, C., Schumann, J., and Mayr, K. (1997). The Model Elimination Provers SETHEO and E-SETHEO. *Journal of Automated Reasoning*, 18:237–246.

Neumann, P. G. (1995). *Computer Related Risks*. ACM Press.

Nonnengart, A., Rock, G., and Weidenbach, C. (1998). On Generating Small Clause Normal Forms. In *Proc. CADE-15*, number 1421 in LNAI, pages 397–411. Springer.

Paulson, L. (1997a). Proving properties of security protocols by induction. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press.

Paulson, L. C. (1994). *Isabelle: A Generic Theorem Prover*, volume 828 of LNCS. Springer.

Paulson, L. C. (1997b). Mechanized proofs of security protocols: Needham-Schroeder with public keys. Technical Report 413, University of Cambridge, Computer Laboratory.

Rajan, S., Shankar, N., and Srivas, M. (1995). An Integration of Model-Checking with Automated Proof Checking. In *Proc. CAV '95*, volume 939 of *LNCS*, pages 84–97. Springer.

Reif, W. (1992). The KIV System: Systematic Construction of Verified Software. In Kapur, D., editor, *Proc. 11th International Conference on Automated Deduction (CADE 11)*, volume 607 of LNAI, pages 753–757. Springer.

Reif, W. (1998). Correct Software for Safety-Critical Systems. invited talk, SPPD meeting during CADE-15.

Reif, W., Schellhorn, G., and Stenzel, K. (1997). Proving system correctness with KIV 3.0. In McCune, W., editor, *Proceedings of the 14th International Conference on Automated deduction (CADE 14)*, volume 1249 of *LNAI*, pages 69–72. Springer.

Satyanarayanan, M. (1987). Integrating Security in a Large Distributed System. Technical Report CMU-CS-87-179, CMU.

Schellhorn, G. and Ahrendt, W. (1998). *The WAM Case Study: Verifying Compiler Correctness for PROLOG with KIV*, chapter III.7, pages 165–194. In (Bibel and Schmitt, 1998). Kluwer.

Schellhorn, G. and Reif, W. (1998). Theorem proving in large theories, chapter III.11. In (Bibel and Schmitt, 1998). Kluwer.

Schumann, J. (1995). SiCoTHEO — Simple Competitive parallel Theorem Provers based on SETHEO. In *Proc. of PPAI'95, Montreal, Ca.*

Schumann, J. (1997). Automatic Verification of Cryptographic Protocols with SETHEO. In *Conference on Automated Deduction (CADE) 14*, LNAI, pages 87–100. Springer.

Schumann, J. (1999a). *Automated Theorem Proving in Software Engineering*. Habilitation, Technische Universität München, Institut für Informatik. in preparation.

Schumann, J. (1999b). Automatische Verifikation von Authentifikationsprotokollen. *KI*. Springer.

Schumann, J. and Fischer, B. (1997). NORA/HAMMR: Making Deduction Based Component retrieval Practical. In *Proc. 12th Conf. on Automated Software Engineering (ASE)*, pages 246–254. IEEE Press.

Schumann, J. and Letz, R. (1990). PARTHEO: a High Performance Parallel Theorem Prover. In Stickel, M. E., editor, *Proc. 10th International Conference on Automated Deduction (CADE 10)*, volume 449 of *Lecture Notes in Computer Science*, pages 40 – 56. Springer.

Schumann, J., Suttner, C., and Wolf, A. (1998). Parallel theorem provers based on SETHEO, chapter II.7, pages 261–290. In (Bibel and Schmitt, 1998). Kluwer.

Slaney, J. (1994). FINDER: Finite domain enumerator. In Bundy, A., editor, *Proc. 12th International Conference Automated Deduction*, volume 814 of *Lecture Notes in Artifical Intelligence*, pages 798–801. Springer.

Smith, D. (1998). Deductive support for software development. invited talk, SPPD meeting during CADE-15.

Spiegel (1998). Aussichten eines Klons. *Der Spiegel*, 18.

Stickel, M., Waldinger, R., Lowry, M., Pressburger, T., and Underwood, I. (1994). Deductive Composition of Astronomical Software from Subroutine Libraries. In *Proc. 12th International Conference on Automated Deduction (CADE 12)*, volume 814 of LNAI, pages 341–355. Springer.

Sutcliffe, G. and Suttner, C., editors. (1997). *Journal Automated Reasoning*, volume 18.

Sutcliffe, G., Suttner, C., and Yemenis, T. (1994). The TPTP Problem Library. In *Proc. 12th International Conference on Automated Deduction (CADE 12)*, volume 814 of LNAI, pages 252–266. Springer.

Suttner, C. and Schumann, J. (1993). Parallel Automated Theorem Proving. In Kanal, L., Kumar, V., Kitano, H., and Suttner, C., editors, *Parallel Processing for Artificial Intelligence I*, pages 209–257. Elsevier.

Suttner, C. B. (1995). *Static Partitioning with Slackness.* DISKI. infix-Verlag.

Syverson, P. F. and van Oorschot, P. (1994). On Unifying Some Cryptographic Protocol Logics. In *Proc. of the IEEE Comp. Soc. Sympos. on Research in Security and Privacy*, pages 14–28.

Tammet, T. (1997). Gandalf. *Journal of Automated Reasoning*, 18(2):199–204.

Weber-Wulff, D. (1993). Selling formal methods to industry. In *FME '93: Industrial-Strength Formal Methods*, volume 670 of *LNAI*, pages 671–678. Springer.

Weidenbach, C., Gaede, B., and Rock, G. (1996). Spass and Flotter version 0.42. In McRobbie, M. A. and Slaney, J. K., editors, *Proc. 13th International Conference Automated Deduction*, volume 1104 of *Lecture Notes in Artifical Intelligence*, pages 141–145. Springer.

Wolf, A. (1998). p-SETHEO: Strategy Parallelism in Automated Theorem Proving. In *Proceedings of 7th International Conference on Analytic Tableaux and Related Methods*, LNAI. Springer.

Wolf, A. and Kmoch, A. (1997). Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung an einem Beispiel aus der Harware-Verifikation. SFB Bericht SFB342/20/97A, Technische Universität München.

Wolfram, S. (1991). *Mathematica–A System for Doing Mathematics by Computer.* Addison-Wesley, Reading, MA, USA, second edition.