Electronic Notes in Theoretical Computer Science 80 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume80.html 6 pages

Formal Methods Research at SICS and KTH: An Overview

Mads Dam [1,2] Lars-Åke Fredlund [2]

Swedish Institute of Computer Science
Box 1263, SE–164 29 Kista, Sweden
{mfd, fred}@sics.se

Dilian Gurov [2]

LECS/IMIT , Royal Institute of Technology (KTH)
KTH Electrum 229, SE–164 40 Kista, Sweden
dilian@imit.kth.se

1 Introduction

The Formal Design Techniques (FDT) Lab at SICS [3] with associated members at the Laboratory of Electronics and Computer Systems (LECS) at the Department of Microelectronics and Information Technology, KTH [4] performs research on theories, tools, and applications of formal methods with particular emphasis on security aspects of distributed systems. The overall focus is on automated and semi-automated methods for program analysis and verification at source and byte code levels, and on the formalisation of (security-related) requirements and policies to which these methods apply. The activities of the lab falls in the following three broad areas: (i) Source and byte code verification based on first-order μ-calculus theorem proving; (ii) Verification of JavaCard applet interactions using call-graph abstractions and compositional techniques; (iii) Formalisation and analysis of security properties, in the areas of information flow control, authorisation, and verification of security protocols.

In this short paper we survey the activities of the groups in these areas.

[1] Supported by the European Office of Aerospace Research and Development, Air Force Office of Scientific Research, Air Force Research Laboratory, under Contract No. F61775-01-C0006.
[2] Supported by the European IST project VerifiCard.
[3] http://www.sics.se/fdt/
[4] http://www.imit.kth.se/lecs/

©2003 Published by Elsevier Science B. V.

2 μ–calculus Based Program Verification

As the programming languages and applications we target in this area are complex, dealing with topics such as concurrency and distribution, no completely automatic verification methods can exist. Instead a semiautomatic approach is adopted, which combines manual reasoning supported by a proof assistant tool with automatic reasoning (model checking). We have explored such an approach in the setting of CCS [6], for the Erlang programming language [9,8], and for JavaCard applets (see the following section). The components of the framework is an operational semantics of the language under study, use of the μ–calculus temporal logic extended with data to formulate correctness requirements, a Gentzen–style proof system for expressing program proofs, and a proof assistant tool to assist in proof development.

Taking the work on Erlang as an example, the basis of the proof system is the standard Gentzen rules for first-order logic, which is extended with rules for fixed point manipulation and the language semantics. The proof system contains a rather complete implementation of a small–step operational semantics for Erlang (including constructs to handle concurrency and error detection and recovery) embedded as a fixed point definition.

Due to the concurrency and dynamism inherent in typical Erlang applications, e.g. [1], a variety of (mutual) induction and co–induction schemes need to be available; at the same time it is often unlikely to foresee which of these might work. We therefore employ symbolic program execution and instance checking to "discover" induction schemes. Our machinery is based on fixed–point ordinal approximation and well–founded ordinal induction, and on a global discharge proof rule for ensuring consistency of the mutual inductions present in a proof structure. The relationship between the global induction discharge rule and standard local fixed point induction rules is explored in Sprenger and Dam [14].

A crucial property of these proof system is the use of compositional reasoning to reduce arguments about an component to arguments about properties of its subcomponents. In the case of CCS, for instance, it is common to apply a proof rule to split a goal involving a parallel composition:

$$\frac{\begin{array}{c}\Gamma \ \vdash \ P_1 : \phi_1,\ \Delta \\ \Gamma \ \vdash \ P_2 : \phi_2,\ \Delta \\ \Gamma,\ \mathbf{x} : \phi_1,\ \mathbf{y} : \phi_2 \ \vdash \ \mathbf{x} \mid \mathbf{y} : \phi,\ \Delta\end{array}}{\Gamma \ \vdash \ P_1 \mid P_2 : \phi,\ \Delta}$$

The rule permits to replace the lower proof goal requiring to prove that $P_1 \mid P_2$ satisfies the μ–calculus formula ϕ with the three new proof goals. In the first two goals the obligation is to prove that the agents P_1 and P_2 satisfies the new "cut-formulas" ϕ_1 and ϕ_2, and the third obligation is to prove that if two arbitrary agents $\mathbf{x}$ and $\mathbf{y}$ satisfy ϕ_1 and ϕ_2, then their parallel composition also satisfy ϕ. Such a proof rule is directly derivable from the standard cut–rule of Gentzen–style proof systems. This is a "weakening" rule that permits to abstract away from the syntax

of the agent under study to consider instead its behavioural properties.

Further information, including downloadable prototype proof assistant tools implementing the proof systems, can found on the world-wide web [5].

3 Verification of Multi–Applet JavaCard Applications

Smart cards provide a secure means for storing and using authentication information and other personal data. They are used in mobile telephony, electronic banking, for keeping health care information, and in many other applications. The growing number of such applications, and the desire to implement partnership programmes giving enhanced business opportunities has lead to the development of multi–application smart cards running on open smart card platforms.

The JavaCard open smart card platform [12] offers, among other features, support for post–issuence loading of applets and inter–applet communication via method calls along shareable interfaces. A firewall mechanism guarantees that only those methods of an applet, which implement a shareable interface, are accessible from outside the applet. This security scheme is not very flexible, since it is static in nature: an applet either has access to an interface method or it hasn't, indiscriminately of the current state. This creates the possibility for illicit applet interactions: certain sequences of method invocations can lead to data flow at unexpected states, resulting in unwanted leak of information to certain parties.

The FDT Lab at SICS is developing within the European VerifiCard project [6], and in collaboration with project partner INRIA Sophia–Antipolis a control–flow based verification technique for checking the absence of illicit applet interactions specific to a given application [2,4]. The technique is based on the abstract notion of control–flow graph, whose behaviour is given in terms of pushdown automata which provide a natural execution model for programs with recursion, and a temporal logic specification language for specifying sets of sequences of method invocations which are deemed to be harmless for the given application. To deal with post–issuence loading, we adopt a compositional approach to verification, allowing global control–flow properties of the whole system to be reduced to local control–flow properties of the individual applets. The latter are then checked using standard methods for model–checking temporal properties of pushdown systems [3]. The correctness of property decompositions is checked automatically following an approach based on maximal models [11].

We are currently assembling a tool set supporting our verification technique. Control–flow graphs are extracted with the help of a static analysis tool based on the SOOT framework [7]. After a property decomposition scheme has been chosen, its correctness is automatically checked through maximal model extraction. Finally, local properties are checked by translating control–flow graphs to push-

[5] `http://www.sics.se/fdt/vericode/`
[6] `http://verificard.org/`
[7] `http://www.sable.mcgill.ca/soot/`

down automata, and using standard tools for temporal logic model–checking of pushdown automata like Moped [7].

The technique and the tool set are being evaluated on several case studies. One of these is a multi–applet electronic purse application called PACAP [8] supporting loyalty programs, which was developed by Gemplus to provide a real case study for researchers working on the JavaCard platform. The owner of such an electronic purse smart card can decide on joining a loyalty program of some company, and load the appropriate applet on her card. Loyalties can establish partnership relations for sharing bonus points. For efficiency reasons, the electronic purse keeps a (circular) log table of bounded size of all credit and debit transactions, and the loyalty applets can request the information stored in this table. When the log table is full, the following transactions will start overwriting already existing entries in the table; to ensure that loyalties do not miss any of the logged transactions they can subscribe (presumably for a fee) to a notification service signalling all subscribed applets every time the log table gets full, so that these can update their local balance. This creates the possibility for illicit applet interactions, since a subscribed loyalty could invoke a non–subscribed partner loyalty – in order to compute an extended balance for instance – giving the latter the possibility to circumvent subscribing to the log–full service. It is the absence of this kind of scenaria which we specify and verify in this case study.

4 Formalisation and Analysis of Security Properties

Within the specific security domain recent work has focused on information flow control, authorisation and delegation, and security protocol analysis using epistemic logics. In this note we focus attention on the area of information flow control.

This problem has received quite some attention in the formal methods community recently, in particular in the area of language-based security (cf. [13]). Most work is based on the multi-level security model in which objects are assigned partially ordered security labels, and for a secure program the task is to show that information does not flow downward in this partial order, from higher security levels to lower ones. Even if most research in the information flow area has been based on the multi-level security model, the problem is that in many applications it is not very useful. The fundamental issue is that many application actually involves some form of intended information leakage. For instance, keys are used for signing and encryption, and pin codes and passwords are used to unlock functionality the existence of which will often be observable to a low-level observer. Unfortunately, these phenomena cause flow of information. Hopefully, the flow is harmless (the design of the protocols, crypto primitives and api's are relied upon to ensure this), but it is flow nonetheless, causing the multi-level security model to break down. Moreover, the problem is not so easily remedied, since the mere presence of an observable high-to-low link can be used by a piece of malicious code to leak

[8] `http://www.gemplus.com/smart/r_d/publications/case-study/`

information at will, using simple bit codings or delays.

To address this we have proposed a notion of *admissible interference* [5,10] based, roughly, on the idea of non-interference up to a tightly controlled use of interface actions which may depend on confidential information. Consider, as an example, confidentiality of the private RSA exponent in a simple implementation of the PKCS#11 cryptographic token standard. Such a token will implement a non-trivial amount of functionality related to pin assignment and use, certificate management, key generation or instantiation, and so on. For confidentiality of the private exponent only a little part of this functionality is relevant, abstractly described in terms of the following little automaton (for the case of off-token key generation):

```
C = while true do begin
      cmd := read(cmdChan) ;
      case cmd of
        assign_private_exponent:
          pExp := read(adminChan) ;
        sign:
          begin mHash := read(userChan);
          write(outChan,signature(mHash,pExp)) end
        ...:  ...
      end
    end
```

This automaton describes the flow of information relevant to confidentiality of the private exponent. Several issues without direct impact on this issue are ignored, including, in particular, authentication. Observe, though, that authentication *is* important to protect against known ciphertext attacks.

The task of a confidentiality analysis using C as an information flow specification is to ensure that the protocol implementation acts according to C and, in particular, does not introduce information dependencies not already allowed by C (for instance by embedding information in the signature). To guard against this we may consider how behaviour of the protocol implementation changes according to changes in the confidential input. In this example, arbitrary changes of input which respects typing (not specified above) should be considered, but this is not always the case. The idea, then, is to reduce confidentiality of a protocol P implementing C, roughly, to a check that (1) the automation C is a correct abstraction of P, and (2) that the behaviour of P is in a suitable sense invariant under reinterpretations f of secret values. We refer to [10] for more details on the approach.

Currently we are working, in the context of the EU IST project VerifiCard[9], on an instantiation of this idea to JavaCard byte code, including a reformulation of the approach which demonstrates more clearly the link to existing security type systems such as that of Volpano and Smith [15].

[9] http://www.verificard.org/

References

[1] Arts, T. and M. Dam, *Verifying a distributed database lookup manager written in Erlang*, in: J. M. Wing, J. Woodcock and J. Davies, editors, *FM'99—Formal Methods, Volume I*, Lecture Notes in Computer Science **1708** (1999), pp. 682–700.

[2] Barthe, G., D. Gurov and M. Huisman, *Compositional verification of secure applet interactions*, in: *Proc. FASE'02*, Lecture Notes in Computer Science **2306** (2002), pp. 15–32.

[3] Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification on infinite structures*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North Holland, 2000 pp. 545–623.

[4] Chugunov, G., L.-Å. Fredlund and D. Gurov, *Model checking of multi-applet JavaCard applications*, in: *Proc. CARDIS'02* (2002), pp. 87–95.

[5] Dam, M. and P. Giambiagi, *Confidentiality for mobile code: The case of a simple payment protocol*, in: *13th IEEE Computer Security Foundations Workshop*, 2000, pp. 233–244.

[6] Dam, M. and D. Gurov, *Compositional verification of CCS processes*, *In* Proc. PSI'99, *Lecture Notes in Computer Science* **1755** (2000), pp. 247–256.

[7] Esparza, J. and S. Schwoon, *A BDD-based model checker for recursive programs*, in: *Proc. of CAV'01*, Lecture Notes in Computer Science **2102** (2001), pp. 324–336.

[8] Fredlund, L., A Framework for Reasoning about Erlang Code, *PhD thesis, Dept. of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm, and Swedish Institute of Computer Science* (2001).

[9] Fredlund, L., D. Gurov, T. Noll, M. Dam, T. Arts and G. Chugunov, *A verification tool for Erlang*, Software Tools for Technology Transfer (2003), accepted for publication.

[10] Giambiagi, P. and M. Dam, *On the Secure Implementation of Security Protocols* (2003), pp. 144–158.

[11] Grumberg, O. and D. E. Long, *Model checking and modular verification*, ACM Transactions on Programming Languages and Systems **16** (1994), pp. 843–871.

[12] *JavaCard 2.1.1 Documentation*, Technical report, Sun Microsystems (2000), `http://java.sun.com/products/javacard/specs.html#211`.

[13] Sabelfeld, A. and A. Myers, *Language-based information-flow security*, IEEE Journal on Selected Areas in Communications **21** (2003), pp. 5–19.

[14] Sprenger, C. and M. Dam, *On the structure of inductive reasoning: Circular and tree-shaped proofs in the mu-calculus*, *To appear in* proc. FOSSACS'03, Warsaw, Poland (2003).

[15] Volpano, D., G. Smith and C. Irvine, *A sound type system for secure flow analysis*, Journal of Computer Security **4** (1996), pp. 1–21.