Where next for formal methods?

James Heather and Kun Wei

Department of Computing, University of Surrey, Guildford, Surrey GU2 7XH, UK
Email: {j.heather, k.wei}@surrey.ac.uk

Abstract. In this paper we propose a novel approach to the analysis of security protocols, using the process algebra CSP to model such protocols and verifying security properties using a combination of the FDR model checker and the PVS theorem prover. Although FDR and PVS have enjoyed success individually in this domain, each suffers from its own deficiency: the model checker is subject to state space explosion, but superior in finding attacks in a system with finite states; the theorem prover can reason about systems with massive or infinite states spaces, but requires considerable human direction. Using FDR and PVS together makes for a practical and interesting way to attack problems that would remain out of reach for either tool on its own.

1 Introduction

Security protocols are vital for providing secure communication and processing of information across a distributed system. However, designing such protocols is notoriously error-prone, since it is difficult to predict and allow for all the possible interactions between the parties operating on the network running the protocol.

Constructing proofs of correctness by hand can be arduous. Indeed, many convincing hand-constructed 'proofs' of correctness of protocols have been published in the literature only to be found wanting at a later date. Over the past decade, formal methods have been remarkably successful in their application to the analysis of security protocols with the emergence of some powerful verification tools.

There are essentially two approaches: model checking and theorem proving. Under a model-checking approach, a system executing the security protocol is represented as a transition system with finitely many states. The model checker then uses various efficient state exploration techniques to discover whether the system can reach a state representing a security violation. Many different model checkers have been employed in this fashion; for example, FDR [7, 5, 10] has proved to be an excellent tool for modelling and verifying safety properties such as authentication and confidentiality. One has to be extremely careful when using a model checker for such tasks, however: it is all to easy to allow the state space to become unmanageably large.

The alternative is the theorem-proving approach, in which a system and its properties are described by logical formulae, and the formal proof is established by proving theorems that state that such properties hold in the system. One successful such setup is that of the rank functions theory [2] embedded in the PVS theorem prover, which can tackle authentication properties of protocols running on networks involving arbitrarily many agents and with an arbitrarily large message space. However, even when using semi-automated (interactive) provers such as PVS or Isabelle, it is a large task to validate a complex system. For example, in the project to verify SET [6], a e-commerce protocol, Isabelle presents the user with subgoals that are hundreds of lines long, and diagnosing a failed proof requires meticulous examination of huge formulae.

The model-checking approach is superior in finding attacks in a system with finite states, but subject to the state explosion problem; the theorem-proving approach can reason about systems with massive or infinite states spaces, but does not provide automatic verification. One natural question to ask is whether it is possible to blend the two complementary approaches in an elegant way to avoid the weaknesses of each.

There have been various lines of investigation for creating hybrid systems. For example, Cohen [1] proposes a proof method for analyzing security protocols in which safety properties are proven by ordinary first-order reasoning, and all proof is generated in an automatic verifier, TAPS. Song [12] also proposes an efficient automatic checking algorithm, Athena, which incorporates its own logic and exploits several state space reduction techniques based on an extension of the Strand Spaces Model [13]. Heather [3] develops a tool, RankAnalyser, that makes use of results [4] to construct a rank function and verify a protocol automatically. It is appropriate for verifying networks of arbitrary size, and with arbitrarily many concurrent executions of the protocol.

However, the above tools are designed for analyzing a few specific properties, all of which are safety properties. Liveness properties—deadlock, non-repudiation, denial of service, and so on—have not yet been mastered to the same degree since they must be expressed in a more complex model. We here propose the novel idea of using the process algebra CSP to describe the system executing the security protocol and the security properties to be verified, and construct the proof of correctness by using a combination of FDR and PVS.

The general approach is to start by modelling the (infinite-state) system in the CSP semantics that we have embedded into PVS, and then start to prove the theorems using PVS. In the course of constructing the proofs, we invariably encounter some subgoals involving only finite-state processes. It would take a long time to trace through the states one by one checking for correspondence in PVS, whereas FDR can verify such cases very quickly; therefore, we proceed by building these results into the PVS theory as axioms, and then proving them correct in FDR. In this way, we harness the power of the theorem prover for establishing results about an infinite-state system, whilst retaining the speed and automation of a model-checker for certain appropriate parts of the proof.

Currently, translating between the PVS syntax and the FDR syntax is done manually; however, we are making progress towards a tool to perform this translation automatically.

2 CSP notation

CSP is an event-orientated language for describing concurrent systems and their interactions. A security protocol is a concurrent system in which a series of messages are exchanged among the various parties involved. CSP is therefore well suited to the modelling and analysis of security protocols.

In CSP, a system can be considered as a process that might be hierarchically composed of many smaller processes. An individual process can be combined with events or other processes by operators such as prefixing, choice, parallel composition, and so on.

Stop is a stable deadlocked process that never performs any events. The process $c \rightarrow P$ behaves like P after performing the event c. A event like c may be compounded; for example, one often-used pattern of events is $c.i.j.m$, consisting of a channel c, a sender i, a receiver j and a message m.

The external choice $P_1 \Box P_2$ may behave either like P_1 or like P_2, depending on what events its environment initially offers it. The traces of internal choice $P_1 \sqcap P_2$ are the same as those of $P_1 \Box P_2$, but the choice in this case is non-deterministic.

The process $P_1 \ {}_A\|_B \ P_2$ is the process where all events in the intersection of A and B must be synchronized, and other events within A and B can be performed independently by P_1 and P_2 respectively. An interleaving $P_1 \ ||| \ P_2$ executes each part entirely independently and is equivalent to $P_1 \underset{\emptyset}{\|} P_2$.

The process $P \setminus A$ will pass through the same events as P, but events in the set A become be invisible. The renamed process $P[a \leftarrow b]$ means that the event a is completely replaced by b in the process P. In addition, processes may also be described recursively whenever such descriptions are well defined.

A trace is defined to be a sequence of finite events. A refusal set is a set of events from which a process can fail to accept anything no matter how long it is offered; $\mathit{refusals}(P/t)$ is the set of P's refusals after the trace t; then (t, X) is a failure in which X denotes $\mathit{refusals}(P/t)$. If the trace t can make no internal progress, this failure is called a *stable failure*.

Liveness is concerned with behaviour that a process is guaranteed to make available, and can be inferred from stable failures; for example, if, for a fixed trace t, we have $a \notin X$ for all stable failures of P of the form (t, X), then a must be available after P has performed t.

Verification in FDR is done by means of determining whether one process refines another. In the stable failures model, this equates to checking whether the traces and failures of one process are subsets of the traces and failures of the other:

$$P \sqsubseteq_F Q \equiv \mathit{traces}(P) \supseteq \mathit{traces}(Q) \wedge \mathit{failures}(P) \supseteq \mathit{failures}(Q)$$

For the properties we are considering, if P meets the properties we are verifying, then Q also meets them if Q refines P. For a fuller introduction, the reader is referred to [8, 11].

3 Case Study

An elegant example to demonstrate the power of the combination of PVS and FDR is the dining philosophers problem. We imagine n philosophers sitting at a table with a bowl of spaghetti in the middle. Between each pair of adjacent philosophers, there is a single fork; and to eat, a philosopher must be holding both of the forks that are beside him. We assume all philosophers pick forks up in the same order—right hand first—and do not put down either fork they have picked up until they have grabbed both. There are various ways in CSP to model this problem, and one of them is as follows:

$$\begin{aligned} PHIL_i =& pickup.i.i \rightarrow pickup.i.i \oplus_n 1 \\ & \rightarrow putdown.i.i \oplus_n 1 \rightarrow putdown.i.i \rightarrow PHIL_i \\ FORK_i =& pickup.i.i \rightarrow putdown.i.i \rightarrow FORK_i \\ & \Box pickup.i \ominus_n 1.i \rightarrow putdown.i \ominus_n 1.i \rightarrow FORK_i \\ COLLEGE =& \Big\|_{i=0}^{n-1} (PandF_i, AP_i) \end{aligned}$$

where '$\oplus_n$' denotes addition modulo n and $PandF_i$ is the combination

$$PHIL_i \underset{AF_i}{\|} FORK_i$$

The alphabet sets used are:

$$\begin{aligned} AF_i &= \alpha PHIL_i \cap \alpha FORK_i \\ AP_i &= (\alpha PHIL_i \cup \alpha FORK_i) \setminus AF_i \end{aligned}$$

Obviously for the dining philosophers problem, the one and only situation causing deadlock is that in which all philosophers hold their right-hand fork simultaneously and wait for their neighbours to put down their forks. There are many modifications one can make to avoid deadlock, one of which results in the asymmetric dining philosophers problem: one philosopher picks up a left-hand fork first.

The basic strategy we adopt is similar to an induction used in [9], where the authors use a hierarchical compression technique in FDR to prove the case with huge numbers of philosophers, but not with arbitrary numbers of philosophers. The key idea is that we can prove that any number greater than one of right-handed pairs of philosophers and forks are equivalent by hiding their internal events and carefully renaming their interface events. The proof starts from the case with $n = 3$ philosophers; then, for the inductive step, we assume that the case of $n = k$ philosophers is deadlock-free, and show that the system remains deadlock-free when the number of philosophers is $n = k + 1$.

Figure 1 shows the dining philosophers network's structure, composed of philosopher/fork pairs. This figure also shows how we deduce deadlock freedom

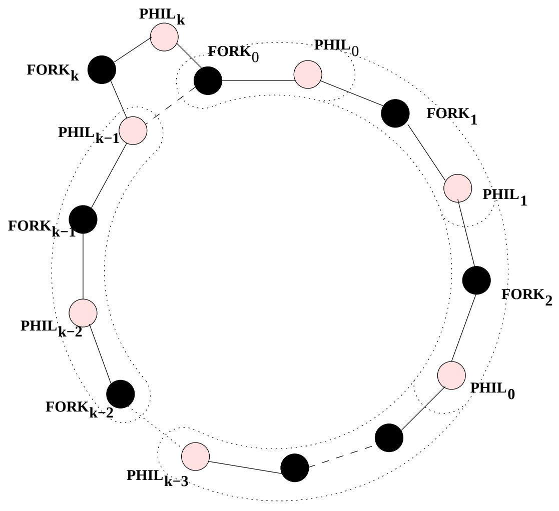

Fig. 1. Inductive structure of dining philosophers

of $k+1$ philosophers from the case of k philosophers. The real thing we want to achieve behind this step is to prove the equivalence of two processes: k philosophers and $k+1$ philosophers. Of course, it is unnecessary to compare all pairs, and we need to concentrate only on the last two pairs in the circle of figure 1. The key to the induction is that if we hide the internal events of the synchronization of $PandF(k, k-2)$ and $PandF(k, k-1)$, it is equivalent to the synchronization of $PandF(k+1, k-2)$, $PandF(k+1, k-1)$ and $PandF(k+1, k)$ with their internal events hidden and $pickup.k.0$ and $putdown.k.0$ renamed as $pickup.(k-1).0$ and $putdown.(k-1).0$ respectively.

The above key lemma is formally described as following:

$$(PandF(k, k-2) \;{}_{AF_{k-2}}\|_{AF_{k-1}}\; PandF(k, k-1)) \setminus IE_k =$$
$$f((\|_{i=k-2}^{k} (PandF(k+1, i), AF_i)) \setminus IE_{k+1})$$

where IE_k and IE_{k+1} denote the sets of internal events and f is a bijective function which renames $pickup.k.0$ and $putdown.k.0$ as $pickup.k-1.0$ and $putdown.k-1.0$ respectively and vice versa.

Although it would be possible to prove this lemma in PVS, it would be in one sense perverse to do so, since it is essentially a very small model-checking exercise. Also proving such a lemma in PVS is rather time-consuming. The natural approach is to establish an axiom for the above lemma and finish the proof in PVS; we transform the script into an FDR script containing an assertion that this lemma holds, and finish the proof using FDR. Using PVS in combination with FDR, then, we can successfully and elegantly prove the asymmetric dining philosophers network with an arbitrary number of philosophers to be deadlock free.

4 Conclusion

Although the example above is not explicitly security-related, we have also found this approach to be highly effective when considering security protocols. For example, we have analyzed and verified the fairness property of the Zhou-Gollmann non-repudiation protocol using a combination of PVS and FDR; this could have been used as the case study, but the dining philosophers example is considerably more transparent, and we considered the digression from the security theme to be a price worth paying for the sake of clarity. The net result of following the PVS/FDR approach is a proof that is automated as far as possible, but that can handle infinite-state systems with minimal effort.

Our aim is to take this work further by automating the translation. We are in the process of developing a tool that can transform PVS scripts into FDR scripts, in order to speed up the process and to avoid introducing unnecessary human error. Ultimately, the procedure will be:

1. model the (infinite-state) system in PVS;
2. use PVS to reduce the proof obligations to finite-state checks;
3. run the translation tool, which will pick up the PVS script and the partially completed proof, and translate the proof obligations into an FDR script containing these obligations as assertions;
4. run FDR on these assertions to complete the proof.

The final stage, we envisage, will involve running a second (far simpler) tool that will run FDR on the resulting script, analyze the results, and insert the proof obligations into the PVS script as axioms for any checks that succeed.

References

1. E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 144–158, Cambridge, UK, 3–5 July 2000. IEEE Computer Society Press.
2. B. Dutertre and S. A. Schneider. Embedding CSP in PVS: an application to authentication protocols. In E. Gunter and A. Felty, editors, *Theorem Proving in Higher-Order Logics: 10th International Conference, TPHOLs '97*, volume 1275 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.
3. J. A. Heather. *'Oh! ... Is it really you?'—Using rank functions to verify authentication protocols.* Department of Computer Science, Royal Holloway, University of London, December 2000.
4. J. A. Heather and S. A. Schneider. Towards automatic verification of authentication protocols on an unbounded network. Technical Report 00-04, Royal Holloway, University of London, 2000.
5. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
6. L. C. Paulson. Verifying the set protocol: Overview. In A. E. Abdallah, P. Ryan, and S. Schneider, editors, *FASec*, volume 2629 of *Lecture Notes in Computer Science*, pages 4–14. Springer, 2002.

7. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. *Proceedings of 8th IEEE Computer Security Foundations Workshop*, 1995.
8. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall International, 1998.
9. A. W. Roscoe, P. H. B. Gardiner, M. Goldsmith, J. R. Hulance, D. M. Jackson, and J. B. Scattergood. Hierarchical compression for model-checking csp or how to check 10^{20} dining philosophers for deadlock. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS*, volume 1019 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 1995.
10. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling And Analysis Of Security Protocols.* Addison Wesley, July 2000.
11. S. A. Schneider. *Concurrent and real-time systems: the CSP approach.* John Wiley & Sons, 1999.
12. D. X. Song. Athena: a new efficient checker for security protocol analysis. *Proceedings of 12th IEEE Computer Security Foundations Workshop*, June 1999.
13. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 1999.