Using the INKA prover to automate safety proofs in abstract interpretation - A case study

Dennis Dams[1], Dieter Hutter[2], and Natalia Sidorova[1]

[1] Dept. of Electrical Eng., Eindhoven University of Technology,
PO Box 513, 5600 MB Eindhoven, The Netherlands
{dennis, natalia}@ics.ele.tue.nl
[2] German Research Center for Artificial Intelligence,
Stuhlsatzenhausweg 3, D-66123, Saarbruecken, Germany
hutter@dfki.de

Automatic verification is a fast growing field with many industrial applications. Model checking is one of the most popular and successful verification techniques. There exists a variety of model-checking tools both in industrial and academic world. In our research, we are mainly oriented on the verification with Spin.

Spin [2] is a software tool that supports the analysis and verification of concurrent systems. The system descriptions are modeled in a high-level language, called Promela. Its syntax is derived from C, and extended with Dijkstra's guarded commands and communication primitives from Hoare's CSP. Given a Promela model as input, Spin generates a C program that performs a verification of the system by enumerating its state space, using a depth-first search algorithm.

A well-known limitation of model checking is its applicability to relatively small finite-state systems only. Introducing property-preserving abstractions is a way to extend these limitations. We use Spin because it proved to be an effective model-checker that is widely used both in academic and industrial world. The essence of building a verification model for a system is abstraction. Only by ignoring details that do not contribute to the correctness property under consideration can one hope to design abstractions that render the automatic verification tractable.

However, the more abstract the verification model is, the harder it will be to relate it to the (specification or implementation of the) original,"real" system. Indeed, as argued in [3], inexperienced users are often inclined to specify too much detail in their Promela models. The reason for this is that such detail is also present in the real system, and by incorporating it in the verification model, the user's confidence in the faithfulness of the model with respect to the real system is increased.

One could say that there exists a trade-off between the effort (by a human user) to relate the model to the real system and the effort (by a model checker) to automatically verify the model. In order to enable efficient automatic verification, the model needs to be as abstract as possible (i.e. as allowed by the property of interest). However, this leaves the user with a considerable task to argue faithfulness. The point is that such an argument is in most cases an informal one. Given the observation that the probability of error for such a task increases with the problem size, the need for a formal and automated approach to the construction of abstractions stands out clearly.

The goal of the research described here is to extend the approach of automated verification to support the construction and justification of abstractions. The key idea of our approach is to provide an extension for Spin to deal with abstractions. The control part of a concrete model would be defined in Promela but all the details on the infinite data type representation and operations on them are moved from it and defined in a language that is an input one for a theorem-prover. Then one can introduce abstractions on the data domain, check the correctness of them (semi-) automatically and afterwards verify the abstract model with Spin. We believe that inductive theorem-provers are more suitable for proofs of safety of abstractions on data domain. The reason for this is that data types and data operations are naturally defined using an inductive style.

To evaluate the applicability of such an approach we considered a model of the Philips' Bounded Retransmission Protocol (BRP) [1]. BRP is a variant of the Alternating-Bit Protocol, where only

* This research has been partially supported by the VIRES project (Verifying Industrial Reactive Systems, Esprit Long Term Research Project #23498).

a bounded number of retransmissions of packets is allowed and time-outs are used to detect packet loss. In the concrete system chunks are lists of natural numbers. That means that the system is infinite. Therefore, we apply data abstraction in order to arrive at a finite verification model.

The essential property to verify is the following: "If the sender is transmitting a list l and the first packet of l arrives at least once at the receiver, then the sequence of packets that the receiving client receives forms a prefix of l".

It can be shown that for the verification of this property it suffices to prove that in a list of non-repeating naturals for any two values e_1 and e_2 on positions i and j resp. in the input list, with $i < j$, one of the following properties holds for the output list:

1. Either (a) e_2 does not occur, or (b) e_1 and e_2 occur in the output list on positions i', j' resp., with $i' < j'$.
2. For any two values e_1 and e_2 on positions i' and j' resp. in the output list, with $i' < j'$, e_1 and e_2 occur on positions i and j resp. in the input list with $i < j$.

The idea of the abstraction we apply is to distinguish two natural numbers which are abstracted into $e1, e2$ resp., and present an arbitrary chunk of naturals by its abstracted head element and abstract tail, where the last one is of the form $epsl, e1l, e2l, e1e2l, e2e1l, error$ ($error$ is an abstraction for a list with the duplicated elements), keeping also the information whether it is an empty or a one-element list.

Though the abstraction seems to be simple, defining even the relatively simple functions on abstract lists is error-prone because of the considerable number of cases that should be taken into account. So we took an attempt to prove the correctness of it with the INKA 4.0 system [4] which is a first-order theorem prover based on a resolution calculus with paramodulation and explicit induction.

It turned out that the arising proof obligations are challenging examples with respect to the speculation of nested case analyses and inductions. On abstract level most of the functions are non-recursively defined. Unfolding a function call by introducing a case analysis allows us to remove this particular occurrence of the function. On the concrete level we are faced with recursively defined functions (operating on inductively defined structures, like lists) and induction is required to explore the nature of these functions. In order to prove the arising theorems, sophisticated case analysis has to be done to unfold non-recursively defined functions which then enables the successful use of the induction rule. Possible case analyses are suggested by the case analyses of the function definitions and by structural information of the used datatypes. To prevent the state explosion problem, only some of these case analyses are combined in order to allow the unfolding of specific function occurrences. Lexicographical orderings based on the definition hierarchy of functions is used to determine whether an occurring subproblem is subject to a further case analysis (or induction respectively) or whether this particular proof attempt is aborted.

Incorporating these improvements into INKA's heuristics for induction and case analysis, we were able to prove the arising proof obligations automatically. While some of the proof obligations can be proved without any user interaction, some of them require the speculation of intermediate (inductive) lemmata which can also be proven automatically by INKA.

References

1. D. Dams, R. Gerth, *The bounded retransmission protocol revisited (extended abstract)*, In Faron Moller, editor, Second International Workshop on Verification of Infinite State Systems (Infinity'97), volume 9 of Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers B.V. (North-Holland), 1997.
2. G. J. Holzmann, *Design and Validation of Communication Protocols*, Prentice Hall, 1991. Also: http://netlib.bell-labs.com/netlib/spin/whatispin.html
3. G. J. Holzmann, *Designing Executable Abstractions. Keynote address*, Proc. Formal Methods in Software Practice, March 1998.
4. D. Hutter, C. Sengler: INKA - The Next Generation. In M. McRobbie J. Slaney (ed), *13th International Conference on Automated Deduction*, Springer Verlag, LNAI 1104, 1996