Using Satisfiability Modulo Theories to Analyze Abstract State Machines

Margus Veanes[1] and Ando Saabas[2⋆]

[1] Microsoft Research, Redmond, WA, USA
margus@microsoft.com
[2] Institute of Cybernetics
Tallinn University of Technology, Tallinn, Estonia
ando@cs.ioc.ee

Abstract. We look at a fragment of ASMs used to model protocol-like aspects of software systems. Such models are used industrially as part of documentation and oracles in model-based testing of application-level network protocols. Correctness assumptions about the model are often expressed through state invariants. An important problem is to validate the model prior to its use as an oracle. We discuss a technique of using Satisfiability Modulo Theories or SMT to perform bounded reachability analysis of such models. We use the Z3 solver for our implementation and we use AsmL as the modeling language.

Protocols are abundant; we rely on the reliable sending and receiving of email, multimedia, and business data. But protocols, such as the Windows network file protocol SMB (Server Message Block), can be very complex and hard to get right. Model programs have proven to be a useful way to model the behavior of such protocols and it is an emerging practice in the software industry [6, 9, 11] to use model programs for documentation and behavioral specification of such protocols, so that different vendors understand the same protocol in the same way. The step semantics of model programs is based on the theory of ASMs [7] with a rich background universe [3]. This enables a range of ASM technologies [4] to be used for analysis of model programs. In the case of model programs, correctness assumptions about the model are often expressed through state invariants. It is important that the model is validated before it is used as a specification or an oracle. We describe a technique of using satisfiability modulo theories or SMT to perform bounded reachability analysis of a fragment of model programs. We use the SMT solver Z3 [5] and we use AsmL [8] as the modeling language. We extend the work in [10] through improved handling of quantifier elimination and extended support for background axioms, in particular bag or multi-set axioms.

The use of SMT solvers for automatic software analysis has recently been introduced [1] as an extension of SAT-based bounded model checking [2]. One advantage of the SMT approach is that it scales better for problems that depend on complex background theories, and the formula for which satisfiability is

⋆ This work was done during an internship at Microsoft Research, Redmond.

checked is quantifier free, rather than propositional. The decision procedure for checking the satisfiability of the formula may use combinations of background theories. The formula is generated after preprocessing of the program. The preprocessing yields a normalized program where all loops have been eliminated by unwinding the loops up to a fixed bound. Unlike traditional sequential programs, model programs operate on a more abstract level and often make use of comprehensions. Moreover, model programs use parallel updates and rich background data structures like sets, maps and bags.

A model program is a finite collection of basic ASMs indexed by actions. The following model program, called *Credits*, is an example of a model program written in AsmL. It specifies how a client and a server need to use message ids, based on a sliding window protocol. Here the client sends requests to the server and the server sends responses back to the client.

```
var window as Set of Integer = {0}
var maxId as Integer = 0
var requests as Map of Integer to Integer = {->}

[Action]
Req(m as Integer, c as Integer)
  require m in window and c > 0
  requests := Add(requests,m,c)
  window := window difference {m}

[Action]
Res(m as Integer, c as Integer)
   require m in requests
   require requests(m) >= c
   require c >= 0
   //require requests.Size > 1 or window <> {} or c > 0 <-- bug
   window := window union {maxId + i | i in {1..c}}
   requests := RemoveAt(requests,m)
   maxId := maxId + c

[Invariant]
ClientHasEnoughCredits()
   require requests = {->} implies window <> {}
```

The *Credits* model program illustrates a typical use of model programs as protocol-specifications. Actions use parameters, maps and sets are used as state variables and a comprehension expression is used to compute a set. Each action has a guard and an update rule given by a basic ASM. For example, the guard of the `Req` action requires that the id of the message is in the current window of available ids and that the number of credits that the client requests from the server is positive. The state invariant associated with the model program is that the client must not starve, i.e. there should always be a message id available at some point, so that the client can issue new requests.

There is a mistake in the model indicated by the missing require-statement. There is a two-action trace leading to a state where the invariant is violated due to this, e.g. the trace `Req(0,1),Res(0,0)`.

There are several different ways of how model programs can be checked for invariant violations. One way is to do explicit state exploration and to use model checking techniques, e.g. this is supported in Spec Explorer [11]. Another approach, that does not require action parameter domains to be provided up-front,

is to represent the bounded reachability problem of the negated invariant as a formula and to check for its satisfiability using a theorem prover.

The *bounded reachability formula* for a given model program P, step bound k and reachability condition φ is:

$$Reach(P, \varphi, k) \quad \stackrel{\text{def}}{=} \quad I_P \wedge \left(\bigwedge_{0 \leq i < k} P[i] \right) \wedge \left(\bigvee_{0 \leq i \leq k} \varphi[i] \right) \tag{1}$$

where I_P is the initial state condition, $P[i]$ is a formula describing step i of the model program, which is an application of some enabled action from the i'th state, and $\varphi[i]$ is φ in the i'th state. The reachability condition φ is typically the negated state invariant. If $Reach(P, \varphi, k)$ is satisfiable, its model can be used to extract an action trace that leads from the initial state to a state violating the invariant. The formula $Reach(P, \varphi, k)$ is typically quantifier free, but involves the use of background theories such as arithmetic, set and multi-set axioms, and map axioms, which makes the use of an SMT solver such as Z3 possible for this kind of analysis.

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In A. Valmari, editor, *SPIN*, volume 3925 of *LNCS*, pages 146–162. Springer, 2006.
2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.
3. A. Blass and Y. Gurevich. Background, reserve, and Gandy machines. In *Proceedings of the 14th Annual Conference of the EACSL on Computer Science Logic*, pages 1–17. Springer, 2000.
4. E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.
5. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'08)*, LNCS. Springer, 2008.
6. W. Grieskamp, D. MacDonald, N. Kicillof, A. Nandan, K. Stobie, and F. Wurden. Model-based quality assurance of windows protocol documentation. In *First International Conference on Software Testing, Verification and Validation, ICST*, Lillehammer, Norway, April 2008.
7. Y. Gurevich. *Specification and Validation Methods*, chapter Evolving Algebras 1993: Lipari Guide, pages 9–36. Oxford University Press, 1995.
8. Y. Gurevich, B. Rossman, and W. Schulte. Semantic essence of AsmL. *Theor. Comput. Sci.*, 343(3):370–412, 2005.
9. J. Jacky, M. Veanes, C. Campbell, and W. Schulte. *Model-based Software Testing and Analysis with C#*. Cambridge University Press, 2008.
10. M. Veanes, N. Bjørner, and A. Raschke. An SMT approach to bounded reachability analysis of model programs. In *FORTE'08*, LNCS. Springer, 2008.
11. M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson. Model-based testing of object-oriented reactive systems with Spec Explorer. In R. Hierons, J. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 39–76. Springer, 2008.