Reasoning about Properties with Abstract State Machines

Gennaro Vessio

Department of Informatics, University of Bari, Bari – 70125, Italy
gennaro.vessio@uniba.it

Abstract. Along the years, Abstract State Machines (ASMs) have been successfully applied for modeling critical and complex systems in a wide range of domains, and for analyzing their computationally interesting properties. However, unlike other well-known and established formalisms, such as Petri nets, they lack of a framework aimed at better supporting their applicability to the formal verification of systems. The goal of my PhD research is to reinforce the ASM formalism as a conceptual tool that developers can find useful and practical in order to analyze properties.

1 Research Objectives

Several formalisms have been successfully applied to the development of critical and complex systems in a wide range of application domains, and to their *ex-ante* and *ex-post* analysis aimed at verifying and validating functionality and quality issues. Representing the system-under-study at a high level of abstraction allows developers to focus on algorithmic aspects, rather than on specific realizations of solutions at lower levels. Moreover, the mathematical foundation of formal methods provides complete and unambiguous investigations about the behavior and the properties the system-under-study is required to exhibit.

In this context, researchers usually distinguish two orthogonal classifications of computationally interesting properties. On one hand, they distinguish between *safety* properties, which stipulate that "something bad" must never happen, and *liveness* properties, which state that "something good" must eventually happen, during computation [23]. On the other hand, they distinguish between *domain-independent* properties, which are interesting for broad classes of systems, and *domain-specific*, which strictly depends on the specific domain [12].

My PhD research is aimed at moving a first step towards the construction of a framework capable to capture computationally interesting properties within the Abstract State Machine (ASM) formalism [13].

1.1 Motivations

Along the years, ASMs have been used for modeling several systems, often concurrent and distributed, and for investigating their properties [13]. However, unlike other well-known and established formalisms, such as finite state machines in model checking [4], Petri nets [22], process algebras (e.g. CSP [21]) and various domain specific languages (DSLs) [12], they lack of a framework aimed at better supporting their applicability to the formal verification of systems.

Concerning domain-independent properties, both safety and liveness, traditional model checking [4] approaches to the problem of verifying properties suffer from two main limitations. First of all, because of decidability issues, they model systems through formalisms, finite state machines or variants, whose expressive power is generally low. Secondly, they require the usage of *declarative* specifications of the properties to be verified, usually through some temporal logic, which are considered by several authors less comfortable for practitioners with respect to *operational* specifications within the same formalism (e.g. [3]). Analogously, when model checking techniques are applied to ASMs, as well as to other general purpose formalisms, these limitations are not overcome (e.g. [16]). Conversely, some formalisms, e.g. Petri nets [22], provide operational, domain-independent characterizations of properties so that the formal verification of the modeled systems can be conducted in a different and sometimes easier manner. However, ASMs lack of these features.

Concerning domain-specific properties, both safety and liveness, of particular interest for my research group is the Mobile Ad-hoc NETwork (MANET) domain [1]. MANETs are wireless networks designed for communications among nomadic hosts: they do not need a physical infrastructure and communications between initiator and destination are established and maintained by the cooperation of the hosts in the network. The twofold role played by hosts (end-point and router), as well as the continuous change of the network topology due to movement, poses problems (study of performance, synchronization issues, and so on), which can benefit from a formal approach. Several DSLs, e.g. CMN [25] and AWN [17], and general purpose formalisms, e.g. finite state machines [14] and Petri nets [6], have been proposed for this purpose. However, these approaches present some drawbacks: understandability, since they are based on a syntax dissimilar to traditional programming languages; expressiveness, because they typically provide only few levels of abstraction; executability, since, especially process algebras, are not directly executable. To my best knowledge, the ASM-based approach has been used in the MANET domain only for specifying the location services of a routing protocol [5]. No work in literature has been devoted to explore the application of ASMs as specific modeling language for MANETs.

1.2 Purposes

On one hand, the goal is to provide operational, domain-independent characterizations of properties in order to overcome the main limitations that penalize traditional model checking techniques applied to ASMs. In fact, the proposed approach can support the properties analysis in such a way that it can be conducted entirely within the ASM framework, so without the need of less expressive models and without the burden of temporal logics. On the other hand, concerning domain-specific properties analysis, the goal is to explore the suitability of ASMs in capturing the specific MANET features. Their use can overcome the various drawbacks suffered by both DSLs and general purpose formalisms typically adopted in this context. Ultimately, the aim is to reinforce the ASM framework as a conceptual tool that developers can find useful and practical for formally analyzing systems properties in critical and complex domains.

With respect to the other formalisms, used both in domain-independent and domain-specific properties analysis, the focus is on ASMs because the advantages they provide under several viewpoints. From the point of view of expressiveness, ASMs represent a general model of computation which “subsumes” all other classic computational models [15]. Indeed, they suffice to capture the behavior of wide classes of sequential [20], parallel [11] and distributed algorithms [18]. Moreover, the ASM approach provides a way to describe algorithmic issues in a simple abstract pseudo-code, which can be translated into a high level programming language source code in a quite simple manner [13]. Secondly, considering methodological issues, the ASM formalism is the basis of a development method which has attracted considerable attention in the last years [13]. Finally, considering the implementation point of view, unlike many other formalisms, ASMs provide a way for translating formal specifications into executable models in order to conduct simulations: this is possible by using tools like CoreASM [16].

It is worth remarking that, since ASMs are Turing-equivalent [20], properties are, in general, undecidable, i.e. their analysis cannot be fully automatized [27]. Nevertheless, in most cases, properties are semi-decidable, so they can still be investigated by focusing on the specific issues of the model under study.

The rest of this paper is organized as follows. Section 2 provides background knowledge on both ASMs and the chosen case study. Section 3 is about the current status of my research. Finally, Section 4 sketches future plans. Since my research is conducted with the support of my supervisor, Alessandro Bianchi, and Sebastiano Pizzutilo, in the following I use the term “we” instead of “I” in order to give credit to their contribution.

2 Background

For better explaining our research, background on both the formalism and the chosen domain is provided. In particular, we take into account the popular Adhoc On-demand Distance Vector (AODV) routing protocol for MANETs [26] as case study for both domain-independent and domain-specific properties analysis.

2.1 Abstract State Machines

ASMs are finite sets of so-called *rules* of the form **if** *condition* **then** *updates* (possibly with the **else** clause) which transform *abstract* states [13]. An ASM state is an algebraic structure, i.e. a domain of objects with functions and relations defined on them. On the other hand, the concept of rule reflects the notion of transition occurring in traditional transition systems: *condition* is a first-order formula whose interpretation can be *true* or *false*, whereas *updates* is a finite set of assignments of the form $f(t_1, \ldots, t_n) := t$, whose execution consists in changing in parallel the value of the specified functions to the indicated value. Pairs of function names, fixed by a signature, together with values for their arguments are called *locations*: they abstract the notion of memory unit. Therefore, a state can be viewed as a function that maps locations to their values: the current configuration of locations together with their values determines the current state of the ASM. As usual in computational models, an ASM *step* is a pair (s, s') of states: in a given state, all conditions are checked, so that all updates in rules whose conditions evaluate to *true* are simultaneously executed, and the result is a transition of the machine from that state to another.

A generalization of basic ASMs is represented by Distributed ASMs (DASMs) [13], capable to capture the formalization of multiple agents acting in a distributed environment. Essentially, a DASM is intended as an arbitrary but finite number of independent agents, each executing its own underlying ASM.

2.2 Ad-hoc On-demand Distance Vector Routing Protocol

AODV is a reactive protocol that discovers and maintains routes on-demand, i.e. routes are built only as desired by initiator nodes using a route request/route reply cycle, which updates routing tables stored in each node [26]. When an initiator needs to start a communication session to a destination, and it does not know a proper route, it broadcasts a route request (RREQ) packet to all its neighbors. An RREQ packet includes information about initiator and destination addresses, their *sequence numbers*, which expresses the freshness of the information, and the length of the route.

Knowledge of routes is stored into routing tables, recorded into a cache memory of each node. A routing table in a node lists all other nodes in the network, and the best (known) route to reach each of them. To this end, each entry of the routing table includes the address of the node, its sequence number, the hop count to reach it, and information about the next node in the route to reach it.

When a node receives an RREQ, it checks if one of the following holds: it is the destination, or destination is one of its neighbors, or it knows a route to destination with corresponding sequence number greater than or equal to the one contained in the RREQ. If so, it unicasts a route reply (RREP) packet back to initiator; otherwise, it rebroadcasts the RREQ. The process is so reiterated until a route to destination is found, or until a previously set timeout expires. While RREP travels towards initiator, routes are set up inside the routing tables of the traversed hosts. Once initiator receives the RREP, communication starts.

3 Current Status of the Research

Currently, my research is dealing with:

- The application of a *predicate abstraction* approach to ASMs, in order to support the verification of both domain-independent and domain-specific properties;
- The definition of *starvation-freedom*, which is a domain-independent liveness property, in terms of ASMs;

– The study of *network topology awareness*, which is a MANET-specific liveness property, by means of ASMs.

In the following, all the above issues are discussed.

3.1 Predicates over ASM States

Classic computational models, such as finite state machines and Turing machines, represent the current state of the computation with (sequence of) symbols belonging to finite alphabets. This poses a limitation: the representation of states is restricted to a specific data structure. Instead, as explained in Section 2.1, ASMs allow any algebraic structure to serve as representation of states. This results in a great amount of details specifying the states, so making the analysis of the properties of the whole system more difficult, mainly for what concerns the comprehension of the semantics of each state, with respect to the computational behavior of the modeled system.

Consider two examples. In the case of a DASM model, the simple execution of one or more updates does not necessarily involve the change of the locations values in such a way that the process makes real computational progress, so driving to starvation. In fact, an ASM could starve even if the computation continues to evolve through different states. In other words, it is difficult to recognize effective progress. On the other hand, the second case concerns, for example, a process that acts both as a client with respect to a service, and, simultaneously, as a server with respect to another service. In this case, ASMs easily capture in a same state different computational activities to be run in parallel. However, it is difficult for the modeler to distinguish, inside the same state, what computational branches have been entered or not.

In order to overcome these problems, the need of an abstraction framework capable to capture the semantics of the ASM states arises. More precisely, there is the need to partition the set of locations into subsets and extract from them the locations specifically interesting for the verification purposes. To this end, we propose a predicate abstraction approach [10]. Predicate abstraction is a popular and widely used technique used to analyze programs [19]. It aims at generating an abstract model from the concrete system to be verified, so checking the former instead of the latter. Briefly speaking, the states of the system are mapped to states of the model according to their evaluation with respect to a finite set of predicates defined over the system states. The model has the same control flow of the original program but it concerns only the predicates over the states.

Literature agrees that a program state coincides with the configuration of program variables together with their current values, e.g. [24]. Analogously, an ASM state coincides with the configuration of ASM locations together with their current values. So, since there exists a natural parallelism between classic program states and ASM states, predicate abstraction can be applied to ASMs as much as to traditional programs. More precisely, it can be applied through the following:

Definition 1. *A predicate ϕ over an ASM state* s *is a first-order formula defined over the locations in* s*, such that* s $\models \phi$.

Predicates over the states serve to represent the semantics of each state, i.e. the properties locally satisfied, and can be regarded as a non-injective labeling function that maps predicates to each state. An ASM model can then be equipped with a set of predicates $\Phi = \{\phi_1, \ldots, \phi_n\}$, such that, in the current state, each ϕ_i can be satisfied or not. In this way, the ASM control flow can be represented by the truth value of the predicates over the states, i.e. by composing the local properties of the various states. So, global properties to be verified can be analyzed by focusing on this composition.

Note that our use of predicate abstraction is quite different with respect to the traditional way: instead of extracting abstract models from the given ASM, the aim is to use predicates over the states in order to support the verification of its properties. In particular, applying predicate abstraction to ASMs induces the partition of locations we need for expressing the semantics of the states.

3.2 ASM-based Starvation-freedom

Literature does not provide a univocal, formal definition of starvation: different authors provide different definitions from different perspectives. In any case, at high description level, starvation can be described as an accident occurring in a multi-process system which hampers a process to continue its proper computation (e.g. [2]). In general, a process starves because it requires the access to an external resource which is never available. In some cases, e.g. in communication systems, the required resource is represented by a message a process waits for.

In [8], we investigated starvation in the AODV routing protocol for MANETs using ASMs. In the paper two formalizations of the protocol are given. The first one is starvation-prone because it intentionally ignores the timeout mechanism adopted for escaping infinite waiting: they arises when routes between initiator and the desired destination lack. Instead, the second one is a refinement which makes the model starvation-free thanks to the reintroduction of the timeout. The comparative analysis of both models suggested that the risk of starvation is due to the presence of what we called *vulnerable rules.*

Definition 2. *A vulnerable rule is a rule of the form* **if** cond **then** update-true **else** update-false *characterized by the following features:*

1. *The truth value of* cond *depends on one or more* risky *functions;*
2. *a) One update between* update-true *and* update-false *generates a computation that does not change the value of a* risky *predicate over the states;*
   b) The computation evolves to a subsequent state, in which the risky predicate over the states does not hold, through the other update.

The functions in feature (1) are *risky* because the risk to starve the system depends on their values. Concerning the class they belong to, it is worth noting that a rigorous classification of ASM functions exists in literature [13]; however, for the purposes of the present work, it is sufficient to remark that they are risky if they represents the dependency of the ASM from external resources. Concerning feature (2a), an important issue is related to the granularity used for defining the states: they are characterized by the same value for the *risky* predicate that expresses the waiting situation [10]. More precisely, if the states are expressed at a finer granularity, then the computation cyclical returns through several intermediate states characterized by the same risky predicate; but if granularity is coarser, then the rule execution could not produce any appreciable change of the ASM state. Moreover, an adequate *stepwise refinement* can be defined so that the case of a single vulnerable rule can be generalized to any finite number of rules, in which at least one rule satisfies the three features above. Finally, it is worth noting that the update allowing the computation to proceed is the "good thing" stated in [23].

The starvation-freedom property does not characterize a single component of a multi-process system, but the system on the whole, so, in order to capture starvation inside the ASM framework, it is worth rising up to the DASM description. In the light of the definition of vulnerable rules, we conjecture that a starvation-free DASM is a DASM composed by *vulnerable rules-free* ASMs.

From a methodological point of view, the vulnerable rules definition implicitly suggests some tasks a modeler can execute for determining the possible presence of starvation risk: analyze the model, looking for cyclical returns to states characterized by the same predicate; if so, the modeler must check if they are driven by conditions depending on some risky function; in this case, the corresponding updates must be studied for investigating their effects. Some of these activities could be supported by automatic tools, such as parsers, dependency graph analyzers, and so on.

3.3 ASMs for Network Topology Awareness in MANETs

In [9], by modeling AODV and proving some computationally interesting properties, we showed that the ASM approach is very useful for capturing the specific MANET features and for reasoning about them. A MANET is characterized by a parallel composition of network nodes in which different sequential activities are, in turn, executed in parallel. The notion of run in a DASM, in which several ASMs are simultaneously executed, and the intrinsic parallel computation of ASM rules allow the modelers to express the nodes' execution in a very natural manner. Secondly, ASM functions, that can be defined over universes of objects of arbitrary complexity, can be used for representing nodes in neighborhood, so abstracting from physical features, and for recording the information about routes stored in routing tables. Thirdly, broadcasting and unicasting of packets can be simply modeled by manipulating abstract messages and by inserting or deleting them into abstract queues. Finally, the similarity between the ASM pseudo-code and the syntax of traditional programming languages provides a way for describing algorithmic issues that developers can find familiar for modeling the system at high abstraction level and for investigating its properties.

Moreover, in [10] we showed how predicate abstraction can help in expressing the node's behavior. In fact, the simultaneous fulfillment of different predicates over the same ASM state is very suitable for capturing the intrinsic concurrency of the nodes' computation.

More specifically, the main focus of this branch of my research is on network topology awareness (NTA). Typically, in reactive protocols, a host has not full knowledge about the network in the whole: it has only a local view, limited to its neighborhood, and information about the other hosts can only be obtained through the dissemination of control packets across the network. This knowledge represents the awareness of each host about the current network topology. In [7], we proposed a variant of AODV aimed at making each host more aware about the current topology. We called the variant N-AODV (NACK-based AODV) because the improvement of NTA is obtained through the introduction of a new control packet: a Not-ACKnowledgement (NACK) packet. In the original AODV, when an intermediate node n receives an instance of an RREQ and does not know a route to reach the desired destination, it simply rebroadcasts the RREQ to all its neighbors. Instead, in N-AODV, in addition to rebroadcasting the RREQ, n unicasts a NACK packet back to initiator. The NACK is so used to inform all nodes between n and initiator that, roughly speaking, n "does not know anything" about the destination. It is worth noting that the usage of NACKs provides information gain from three points of view:

– When a route to destination is found, initiator is aware about not only the next hop in the route to reach destination, but also about all the intermediate nodes of that route. Note that, even if the route discovery process does not end with success, initiator is aware about all nodes reached by an RREQ until the timeout expiration;
– Initiator is also aware about all nodes reached by an RREQ but not necessarily involved in a route to reach destination;
– Because of the forwarding activities due to NACKs unicasting, all nodes in the reverse route to reach initiator, are aware about the senders of the various NACKs.

In the work we described the proposal and proved its correctness.

4 Future Work

The aim of my PhD research is to move a step towards the construction of a framework capable to capture systems properties inside the ASM formalism. To this end, my research develops over two parallel directions: on one hand, the aim is to provide operational characterizations of domain-independent properties in terms of ASMs; in particular, the focus is on starvation-freedom. On the other hand, the aim is to show the suitability of ASMs in modeling specific issues of MANETs; in particular, the focus is on network topology awareness.

Concerning domain-independent properties analysis, the results obtained from the study of starvation in the AODV protocol are encouraging for the purposes of our research. In fact, the operational definition of vulnerable rules allows modelers to treat starvation analysis entirely within the ASM framework, i.e. without translating ASMs into less expressive models and without adopting temporal logics. In this way, the main limitations that penalize traditional model checking techniques applied to ASMs can be overcome. The research will continue with the purpose of generalizing the finding of the case study in order to formally prove the necessary and sufficient conditions that enable starvation inside ASMs. To achieve this goal, we will deepen into the relationship between the syntactic notion of starvation inside ASMs and the classic, semantic notion of starvation in literature.

Conversely, concerning domain-specific properties analysis, ASMs have shown to be very useful for modeling MANETs and investigating their properties. Indeed, the advantages they provide overcome the various drawbacks that affect the other formalisms typically applied in this context, i.e. the lack of understandability, expressiveness and executability features. Moreover, ASMs helped in formally specifying a variant of the AODV protocol, namely N-AODV, aimed at improving the network topology awareness of each node. Since the use of NACKs injects overhead in the computation activities carried on by hosts, the research will continue with the aim to compare AODV and N-AODV performance through simulations in order to evaluate if the overhead is adequately balanced by the information gain obtained. For this purpose, CoreASM provides an excellent simulation environment for conducting experiments.

More in general, in order to support the ASM-based properties analysis, independently from the class a property belongs to, we discovered that predicates over the states provide the abstraction framework we need for managing the high complexity of the representation of the states implicitly given by ASMs.

It is worth noting that the research conducted so far has dealt only with liveness properties. However, future directions could explore more in deep safety properties, both domain-independent and domain-specific. For example, deadlock-freedom as an instance of safety domain-independent properties, and some security issues in MANETs, as instances of safety domain-specific properties.

References

1. Agrawal, D.P., Zeng, Q.A.: Introduction to Wireless and Mobile Systems. Thomson Brooks/Cole (2003)
2. Alpern, B., Schneider, F.B.: Defining Liveness. Information Processing Letters, 21(4), 181–185 (1985)
3. Arcaini, P., Gargantini, A., Riccobene, E.: CoMA: Conformance Monitoring of Java Programs by Abstract State Machines. In: 2nd International Conference on Runtime Verification, 223–238 (2012)
4. Baier, C., Katoen, J.P.: Principles of Model Chacking. The MIT Press (2008)
5. Benczur A, Glässer U, Lukovskzi T.: Formal Description of a Distributed Location Service for Mobile Ad-hoc Networks. In: Börger, E., Gargantini, A., Riccobene, E. (eds), Abstract State Machines 2003 - Advances in Theory and Applications, 2589, 204–217 (2003)

6. Bianchi, A., Pizzutilo, S.: Studying MANET through a Petri Net-Based Model. In: 2th International Conference of Evolving Internet, 220–225 (2010)
7. Bianchi, A., Pizzutilo, S., Vessio, G.: Preliminary Description of NACK-based Ad-hoc On-Demand Distance Vector Routing Protocol for MANETs. In: 9th International Conference on Software Engineering and Applications, 500–505 (2014)
8. Bianchi, A., Pizzutilo, S., Vessio, G.: Starvation Analysis with Abstract State Machines: the AODV Case Study. Informatica - An International Journal of Computing and Informatics (2014) (under review)
9. Bianchi, A., Pizzutilo, S., Vessio, G.: Suitability of Abstract State Machines for Discussing Mobile Ad-hoc Networks. Global Journal of Advanced Software Engineering, 1, 29–38 (2014)
10. Bianchi, A., Pizzutilo, S., Vessio, G.: Applying Predicate Abstraction to Abstract State Machines. In: 20th International Conference on Exploring Modelling Methods for Systems Analysis and Design (2015) (to appear)
11. Blass, A., Gurevich, Y.: Abstract State Machines Capture Parallel Algorithms. ACM Transactions on Computational Logic, 4(4), 578–651 (2003)
12. Bodevaix, J.P., Filali, M., Lawall, J., Muller, G.: Formal Methods Meet Domain Specific Languages. In: 5th International Conference on Integrated Formal Methods, 187–206 (2005)
13. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
14. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized Verification of Ad Hoc Networks. In: 21th Int. Conf. on Concurrency Theory, 313–327 (2010)
15. Dershowitz, N.: The Generic Model of Computation. Electronic Proceedings in Theoretical Computer Science (2013)
16. Farahbod, R., Glässer, U., Ma, G.: Model Checking CoreASM Specifications. In: 14th International ASM Workshop (2007)
17. Fehnker, A., Van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A Process Algebra for Wireless Mesh Networks. In: 21st European Symposium on Programming, 295–315 (2012)
18. Glausch, A., Reisig, W.: An ASM-Characterization of a Class of Distributed Algorithms. Rigorous Methods for Software Construction and Analysis, 50–64, Springer-Verlag (2009)
19. Graf, S., Saidi, H.: Construction of Abstract State Graphs with PVS. In: 9th International Conference on Computer Aided Verification, 72–83 (1997)
20. Gurevich, Y.: Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic, 1(1), 77–111 (2000)
21. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985)
22. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer, 9(3-4), 213–254 (2007)
23. Kindler, E.: Safety and Liveness Properties: A Survey. EATCS Bulletin, 53, 268–272 (1994)
24. Laplante, P.: Dictionary of Computer Science, Engineering and Technology. CRC Press (2000)
25. Merro, M.: An Observational Theory for Mobile Ad Hoc Networks. Information and Computation , 207(2), 194–208 (2009)
26. Perkins, C.E., Belding-Royer, E.M., Das, S.R.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, `http://tools.ietf.org/html/rfc3561` (2003)
27. Spielmann, M.: Automatic Verification of Abstract State Machines. In: 11th International Conference on Computer Aided Verification, 431–442 (1999)