Model Checking Abstract State Machines and Beyond

Marc Spielmann

Mathematische Grundlagen der Informatik,
RWTH Aachen, D-52056 Aachen, Germany
spielmann@informatik.rwth-aachen.de

Abstract. We propose a systematic investigation of the (semi-) automatic verifiability of ASMs. As a first step, we put forward two verification problems concerning the correctness of ASMs and investigate the decidability and complexity of both problems.

1 Introduction

Abstract state machines (ASMs) [Gur95, Gur97, Gur99] have become the formal foundation of a successful methodology for specification and verification of complex hardware and software systems. This is particularly witnessed by numerous publications using the ASM formalism for rigorous mathematical correctness proofs of large-scale applications. (See the recent bibliography [BH98] and the web site [Web]. For an introduction to the ASM verification method the reader is referred to [Bör95].) Interestingly, most of these contributions focus on *manual* verification, while the number of publications where all or part of the verification process is *automated* is rather small. (For exceptions see [Win97, Win00, GR00, BS00] and consult [Web].) In a nutshell, computer-aided verification of ASMs, i.e., (semi-) automatic verification of dynamic systems expressed in terms of ASMs, has not yet been well developed. In view of the success of the ASM verification method in manual verification we think there is need for a *systematic investigation* of the (semi-) automatic verifiability of ASMs. The present paper can be viewed as an attempt to initiate such an investigation.

As a first step toward a systematic investigation of the verifiability of ASMs, we have to make precise what we actually mean by "verifying ASMs". In its full generality, the problem of verifying ASMs can be seen as a *decision problem* of the following kind:

> Given an ASM M (i.e., a formal description of some dynamic system) and a specification φ (i.e., a formal description of a desirable property of the system), decide whether M 'satisfies' φ.

One of our main goals in this paper is to identify decision problems of the above kind such that solving these problems coincides with proving properties of ASMs. We put forward two such problems, which we call the *model-checking problem* and the *verification problem* for ASMs.

The *model-checking problem* for ASMs, denoted MC, can be stated informally as follows:

MC: Given an ASM M, a specification φ, and an input $\mathcal{I}$ appropriate for M, decide whether φ holds during all possible computations of M on $\mathcal{I}$.

Note that, in general, MC cannot be solved by means of testing, i.e., by simply running a given ASM M on a given input $\mathcal{I}$ and observing whether a given specification φ is satisfied. There are two reasons for this. Firstly, M may be non-deterministic or may access external functions or relations. For instance, if M chooses a natural number n in the first step and then proceeds depending on the choice of n, one would have to run M for each of the infinitely many possible choices for n. Secondly, even if M is deterministic and does not interact with its environment, it may not halt on $\mathcal{I}$.

The above formulation of MC immediately raises the question of a *specification language* for ASMs, i.e., a formal language suitable to express properties of ASMs. Since in the literature there is no consensus on the choice of such a language, we advocate here *first-order branching temporal logic* (FBTL) as specification language for ASMs. (FBTL is a straightforward extension of the well-known propositional branching-time logic CTL* by first-order reasoning [Eme90]. For details see the next section.) Another open issue is the notion of *input*. The question is which type of input is suitable for ASMs and how does the initial state of an ASM on some input look like? Concerning this question we follow [BG97, BGS99, GS99, BGdB00] and consider ASMs whose inputs are finite structures. To not impose unnecessary restrictions on the initial states of ASMs, we associate with every ASM M a function that maps every input $\mathcal{I}$ appropriate for M to an initial state of M on $\mathcal{I}$. In principle, this function can be any mapping from inputs to states.

An investigation of MC is motivated mainly by applications where it suffices to ensure correctness of an ASM for only a small number of inputs. As an example, consider an ASM M running on the notebook computer of a salesperson. M receives as input a database $\mathcal{D}$, say, the catalog of a supplier stored on a CD-ROM, and interacts with the salesperson via some external relations. The salesperson may inquire about the availability and prices of certain products or may store customer orders in a dynamic relation of M. Since in this scenario M runs on the given database $\mathcal{D}$ only, we can check whether M satisfies a specification φ by deciding whether $(M, \varphi, \mathcal{D})$ is a positive instance of MC.

Although a solution of MC is interesting for certain applications, there are many applications where correctness of an ASM must be guaranteed for *all* admissible inputs, of which there are often infinitely many. For instance, a compiler is usually supposed to correctly translate an input program of arbitrary length. In such cases one has to to check whether for a given ASM M and a given specification φ, $(M, \varphi, \mathcal{I})$ is a positive instance of MC for *every* admissible input $\mathcal{I}$. This problem, which we call the *verification problem* for ASMs and denote by VERIFY, can be stated informally as follows:

VERIFY: Given an ASM M and a specification φ, decide whether for *every* input $\mathcal{I}$ appropriate for M, φ holds during all possible computations of M on $\mathcal{I}$.

Not surprisingly, both MC and VERIFY are—in their full generality—undecidable, as ASMs are computationally complete. (Indeed, every Turing machine can be regarded as a particularly simple ASM. It is an easy exercise to define a reduction of the halting problem for Turing machines to both problems.) Therefore, we investigate the decidability of MC and VERIFY with respect to *classes of restricted ASMs*. The idea is to formulate conditions on ASMs such that MC (resp. VERIFY) becomes decidable for ASMs that satisfy these conditions. Once decidability with respect to a class of ASMs is established, all ASMs in this class are automatically verifiable in the sense that there exists a procedure which checks whether a given ASM satisfies a given FBTL specification on a particular input (resp. on all inputs).

It is worth noticing that for unrestricted ASMs, MC and VERIFY collapse to the same problem. For example, in order to decide whether (M, φ) is a positive instance of VERIFY, it suffices to decide whether $(M', \varphi, \mathcal{I})$ is a positive instance of MC, where M' ignores $\mathcal{I}$, non-deterministically chooses some other input $\mathcal{I}'$ (of arbitrary size!), and then simulates M on $\mathcal{I}'$. However, MC and VERIFY do not necessarily coincide for restricted ASMs. Consider, for example, ASMs defined by means of a single rule of the form (`if` γ `then` *accept*), where the guard γ is a first-order sentence. Checking whether such an ASM accepts a particular input structure can be done in PSPACE, whereas verifying that it accepts all input structures is undecidable by Trakhtenbrot's theorem [EF95].

Although deciding MC appears to be easier than deciding VERIFY, it is a priori not clear whether a solution of VERIFY (for a class of ASMs) implies a solution of MC (for the same class of ASMs). The question is whether MC is a subproblem of VERIFY in the sense that there exists a reduction of MC to VERIFY with respect to fixed classes of ASMs. We give some formal evidence that this is indeed the case, thereby also providing a justification for the title of the paper.

Outline. In the next section, we recall first-order branching temporal logic which will serve here as specification language for ASMs. In Section 3, we formally define the model-checking problem for ASMs and present some results concerning its decidability and complexity (with respect to classes of restricted ASMs). In Section 4, we define the verification problem for ASMs, show that it subsumes the model checking problem, and study its decidability and complexity.

2 A Specification Language for Abstract State Machines

In order to investigate the automatic verifiability of ASMs we need to fix a formal language suitable to express properties of ASMs. We propose to specify properties of ASMs in terms of first-order branching temporal logic (FBTL). Formally, FBTL formulas express properties of *computation graphs* of ASMs. These graphs can be viewed as collections of runs of ASMs and are introduced next.

2.1 Computation Graphs

We assume the reader to be familiar with (sequential) ASMs [Gur97, Gur95]. To simplify the definition of computation graphs, we first introduce some additional terminology and notation.

ASM vocabularies. An *ASM vocabulary* Υ is a quadruple $(\Upsilon_{\rm in}, \Upsilon_{\rm stat}, \Upsilon_{\rm dyn}, \Upsilon_{\rm ext})$ of finite, pairwise disjoint vocabularies. The symbols in $\Upsilon_{\rm in}$, $\Upsilon_{\rm stat}$, $\Upsilon_{\rm dyn}$, and $\Upsilon_{\rm ext}$ are called *input*, *static*, *dynamic*, and *external symbols*, respectively. Sometimes we also denote by Υ the (ordinary) vocabulary $\Upsilon_{\rm in} \cup \Upsilon_{\rm stat} \cup \Upsilon_{\rm dyn} \cup \Upsilon_{\rm ext}$. The intended meaning will be clear from the context. For technical reasons we shall always assume that $\Upsilon_{\rm in}$ contains the constant symbol 0.

ASM programs. Let $\Upsilon = (\Upsilon_{\rm in}, \Upsilon_{\rm stat}, \Upsilon_{\rm dyn}, \Upsilon_{\rm ext})$ be an ASM vocabulary. An *ASM program* Π over Υ is a rule in the sense of [Gur97], each of whose static (resp. dynamic, external) symbols occurs in $\Upsilon_{\rm in} \cup \Upsilon_{\rm stat}$ (resp. $\Upsilon_{\rm dyn}$, $\Upsilon_{\rm ext}$). Intuitively, the symbols in $\Upsilon_{\rm stat}$ denote those relations and functions which are static and not included in the input of an ASM (e.g., arithmetical operations).

States. A *state* $\mathcal{S}$ over an ASM vocabulary Υ is a structure over the (ordinary) vocabulary Υ. We consider both finite and infinite states of ASMs.

Transitions. Let Π be an ASM program over Υ. The one-step semantics of Π defines a binary *transition relation* $T_{(\Upsilon,\Pi)}$ between states over Υ as usual. That is, $(\mathcal{S}, \mathcal{S}') \in T_{(\Upsilon,\Pi)}$ iff

1. $\mathcal{S}$ and $\mathcal{S}'$ are states over Υ, and
2. $\mathcal{S}'$ is the successor state (or sequel) of $\mathcal{S}$ with respect to Π in the sense of [Gur97].

Remark 1. (a) If Π contains `choose`, `import`, or external symbols, then $T_{(\Upsilon,\Pi)}$ is in general non-deterministic, i.e., there may exist states $\mathcal{S}, \mathcal{S}', \mathcal{S}''$ with $(\mathcal{S}, \mathcal{S}')$, $(\mathcal{S}, \mathcal{S}'') \in T_{(\Upsilon,\Pi)}$ and $\mathcal{S}' \neq \mathcal{S}''$.

(b) If $\mathcal{S}$ is finite and Π contains `import`, then it may happen that Π attempts to import a new element, although the reserve of $\mathcal{S}$ is empty. One possible solution to this problem is to modify the semantics of `import` so that in states with empty reserve it always 'imports' the element denoted by 0. □

Inputs. An *input* $\mathcal{I}$ over an ASM vocabulary Υ is a finite structure over $\Upsilon_{\rm in}$.

Initialization mappings. An *initialization mapping initial* over an ASM vocabulary Υ is a function which maps every input $\mathcal{I}$ over Υ to a state $\mathcal{S}_{\mathcal{I}}$ over Υ satisfying the following three conditions:

1. The universe of $\mathcal{I}$ is a subset of the universe of $\mathcal{S}_{\mathcal{I}}$.
2. For every relation symbol $R \in \Upsilon_{\rm in}$, every k-ary function symbol $f \in \Upsilon_{\rm in}$, and every k-tuple $\bar{a}$ of elements in $\mathcal{S}_{\mathcal{I}}$,

$$R^{\mathcal{S}_{\mathcal{I}}} = R^{\mathcal{I}} \quad \text{and} \quad f^{\mathcal{S}_{\mathcal{I}}}(\bar{a}) = \begin{cases} f^{\mathcal{I}}(\bar{a}) & \text{if } \bar{a} \text{ consists of elements in } \mathcal{I} \\ 0^{\mathcal{I}} & \text{otherwise.} \end{cases}$$

3. For every input $\mathcal{I}'$ over Υ, if $\mathcal{I}'$ and $\mathcal{I}$ are isomorphic, so are $\mathcal{S}_{\mathcal{I}'}$ and $\mathcal{S}_{\mathcal{I}}$.

Abstract state machines. An *abstract state machine* M is a triple (Υ, *initial*, Π) consisting of

- an ASM vocabulary Υ,
- an initialization mapping *initial* over Υ, and
- an ASM program Π over Υ without free variables.

Υ_{in}, Υ_{stat}, Υ_{dyn}, and Υ_{ext} are called the *input*, *static*, *dynamic*, and *external vocabulary* of M, respectively. An *input* $\mathcal{I}$ appropriate for M is an input over Υ. M is *deterministic* if Π does not contain `choose`.

We can now define the notion of a computation graph of an ASM.

Definition 2. Let $M = (\Upsilon, \mathit{initial}, \Pi)$ be an ASM and $\mathcal{I}$ an input appropriate for M. The *computation graph* of M on $\mathcal{I}$, denoted $C_M(\mathcal{I})$, is a triple $(\mathit{States}, \mathit{Trans}, \mathcal{S}_0)$ where

- *States* is the set of those states over Υ whose universe is identical with the universe of $\mathit{initial}(\mathcal{I})$,
- $\mathit{Trans} \subseteq \mathit{States} \times \mathit{States}$ is the restriction of $T_{(\Upsilon,\Pi)}$ to *States*, and
- $\mathcal{S}_0 := \mathit{initial}(\mathcal{I})$.

A *run* of M on $\mathcal{I}$ is an infinite path in $C_M(\mathcal{I})$ starting at $\mathcal{S}_0$. □

Notice that for every ASM $M = (\Upsilon, \mathit{initial}, \Pi)$ and for every input $\mathcal{I}$ appropriate for M, $C_M(\mathcal{I})$ is finite iff $\mathit{initial}(\mathcal{I})$ is finite.

Example 3. In this and the next example (Example 5) we show that classical model checking, i.e., checking whether a given finite-state system is a model of a given temporal formula (see, e.g., [CES86]), is a special case of model checking ASMs. To this end, we display below a simple ASM M whose computation graphs are finite-state systems.

In classical model checking, finite-state systems are often represented as Kripke structures. A *Kripke structure* $\mathcal{K}$ is a tuple

$$(S, T, s_0, P_1, \ldots, P_k) \tag{1}$$

consisting of a finite set S of states, a binary transition relation $T \subseteq S \times S$, an initial state $s_0 \in S$, and a sequence $P_1, \ldots, P_k$ of subsets of S. (Sometimes, the sequence $P_1, \ldots, P_k$ is also given in the form of a labeling function $L : \{p_1, \ldots, p_k\} \to 2^S$ with $L(p_i) = P_i$ for each $i \in \{1, \ldots, k\}$.) Since it is customary in the context of model checking, we assume that every $s \in S$ has at least one T-successor, which may be s itself. $\mathcal{K}^- := (S, T, s_0)$ is called the *transition graph* of $\mathcal{K}$.

We define $M = (\Upsilon, \mathit{initial}, \Pi)$ so that for every Kripke structure $\mathcal{K}$, the computation graph of M on input $\mathcal{K}^-$ is isomorphic to $\mathcal{K}^-$. The vocabulary of M is defined as follows: $\Upsilon_{\text{in}} := \{T, 0\}$, where T is a binary relation symbol (which will be interpreted as the edge relation of the input transition graph), $\Upsilon_{\text{dyn}} := \{\mathit{pebble}\}$, where *pebble* is a nullary function symbol, $\Upsilon_{\text{stat}} := \varnothing$, and $\Upsilon_{\text{ext}} := \varnothing$. The program of M is:

```
program Π:
  choose x : T(pebble, x)
      pebble := x
```

Observe that the transition graph $\mathcal{K}^-$ of any Kripke structure $\mathcal{K}$ is a finite structure over Υ_{in}, and thus an input over Υ. The initialization mapping of M is defined by $\mathit{initial}(\mathcal{K}^-) := (\mathcal{K}^-, 0^{\mathcal{K}^-})$, where $(\mathcal{K}^-, 0^{\mathcal{K}^-})$ denotes the state over Υ in which T and 0 are interpreted as in $\mathcal{K}^-$, and the nullary dynamic function symbol *pebble* is interpreted as $0^{\mathcal{K}^-}$. One can now verify that $\mathcal{S} \mapsto \mathit{pebble}^{\mathcal{S}}$ is an isomorphism between $C_M(\mathcal{K}^-)$ and $\mathcal{K}^-$. □

2.2 First-Order Branching Temporal Logic

Recall that all runs of an ASM M on a particular input $\mathcal{I}$ are embedded in the computation graph of M on $\mathcal{I}$. Hence, it is reasonable to express a property of M (on any input) as a property of *all* computation graphs of M. To express properties of computation graphs we propose first-order branching temporal logic (FBTL), which is a combination of first-order logic and the well-known propositional branching temporal logic CTL* [CES86, Eme90]. On the one hand, first-order logic allows us to reason about the states of an ASM. (Recall that the states of an ASM are first-order structures.) On the other hand, CTL* enables us to reason about the temporal and non-deterministic behavior of an ASM. For instance, one can express that 'good things eventually happen' or that 'there exists a good run' of an ASM. For the reader's convenience we recall the definition of FBTL here.

Definition 4. *State formulas* and *path formulas* of *first-order branching temporal logic* are defined by simultaneous induction:

- **(S1)** Every atomic (first-order) formula is a state formula.
- **(S2)** If φ is a path formula, then $\mathbf{E}\varphi$ is a state formula.
- **(P1)** Every state formula is a path formula.
- **(P2)** If φ and ψ are path formulas, then $\mathbf{X}\varphi$, $\varphi\mathbf{U}\psi$, and $\varphi\mathbf{B}\psi$ are path formulas.
- **(SP1)** If φ and ψ are state (resp. path) formulas, then $\varphi \vee \psi$ and $\neg\varphi$ are state (resp. path) formulas.
- **(SP2)** If x is a variable and φ a state (resp. path) formula, then $\exists x\varphi$ is a state (resp. path) formula.

The *free* and *bound variables* of state and path formulas are defined in the obvious way. FBTL denotes the set of state formulas. □

Semantics of FBTL formulas. Intuitively, a state formula of the form $\mathbf{E}\varphi$ expresses that there (E)xists an infinite path (presumably a run of an ASM) such that the path formula φ holds along this path. The intuitive meaning of path formulas of the form $\mathbf{X}\varphi$, $\varphi\mathbf{U}\psi$, and $\varphi\mathbf{B}\psi$ is as follows:

- $\mathbf{X}\varphi$: φ holds in the ne(X)t state.

– $\varphi\mathbf{U}\psi$: ψ holds eventually and φ holds (U)ntil then.
– $\varphi\mathbf{B}\psi$: either ψ holds always or φ holds (B)efore ψ fails.

We formally define the semantics of FBTL formulas only with respect to computation graphs of ASMs. Let Υ be an ASM vocabulary, φ a state formula over Υ, ψ a path formula over Υ, and $C = (\mathit{States}, \mathit{Trans}, \mathcal{S}_0)$ a computation graph of an ASM of vocabulary Υ. W.l.o.g., we may assume that $\mathrm{free}(\varphi) = \mathrm{free}(\psi) = \{\bar{x}\}$. Let $\rho = (\mathcal{S}'_i)_{i\in\omega}$ be an infinite path in C (not necessarily starting at the initial state $\mathcal{S}_0$ of C). For any $j \in \omega$, let $\rho|j$ denote the infinite path $(\mathcal{S}'_{i+j})_{i\in\omega}$, i.e., the suffix $\mathcal{S}'_j, \mathcal{S}'_{j+1}, \ldots$ of ρ.

Simultaneously for every state $\mathcal{S}$ in C, every infinite paths ρ in C, and all interpretations $\bar{a}$ of the variables $\bar{x}$ (chosen from the universe of $\mathcal{S}_0$) define the two *satisfactory relations* $(C, \mathcal{S}, \bar{a}) \models \varphi$ and $(C, \rho, \bar{a}) \models \psi$ by induction on the construction of φ and ψ:

(S1)	$(C, \mathcal{S}, \bar{a}) \models \varphi$	$:\Leftrightarrow$	$\mathcal{S} \models \varphi[\bar{a}]$, where φ is an atomic formula
(S2)	$(C, \mathcal{S}, \bar{a}) \models \mathbf{E}\varphi$	$:\Leftrightarrow$	there is an infinite path ρ' in C starting at $\mathcal{S}$ such that $(C, \rho', \bar{a}) \models \varphi$
(P1)	$(C, \rho, \bar{a}) \models \varphi$	$:\Leftrightarrow$	$(C, \mathcal{S}'_0, \bar{a}) \models \varphi$, where $\mathcal{S}'_0$ is the first state of ρ
(P2)	$(C, \rho, \bar{a}) \models \mathbf{X}\varphi$	$:\Leftrightarrow$	$(C, \rho\vert 1, \bar{a}) \models \varphi$
	$(C, \rho, \bar{a}) \models \varphi\mathbf{U}\psi$	$:\Leftrightarrow$	there exists $i \in \omega$ such that $(C, \rho\vert i, \bar{a}) \models \psi$ and for all $j < i$, $(C, \rho\vert j, \bar{a}) \models \varphi$
	$(C, \rho, \bar{a}) \models \varphi\mathbf{B}\psi$	$:\Leftrightarrow$	for every $i \in \omega$, if $(C, \rho\vert i, \bar{a}) \models \neg\psi$, then there exists $j < i$ with $(C, \rho\vert j, \bar{a}) \models \varphi$

The semantics of formulas derived by means of rule **(SP1)** is standard. It remains to declare the semantics of formulas derived by means of rule **(SP2)**. Below, σ stands for either a state $\mathcal{S}$ or an infinite path ρ in C, depending on whether φ is a state or a path formula.

(SP2) $(C, \sigma, \bar{a}) \models \exists y\varphi(\bar{x}, y) \quad :\Leftrightarrow$ there is an element b in the universe of $\mathcal{S}_0$ such that $(C, \sigma, \bar{a}, b) \models \varphi(\bar{x}, y)$

For every FBTL sentence φ over Υ, let

$$C \models \varphi \quad :\Leftrightarrow \quad (C, \mathcal{S}_0) \models \varphi.$$

The following abbreviations are customary and will be used frequently:

– $\mathbf{A}\varphi := \neg\mathbf{E}\neg\varphi$ (φ holds along every path).
– $\mathbf{F}\varphi := \mathit{true}\,\mathbf{U}\varphi$ (φ holds eventually).
– $\mathbf{G}\varphi := \mathit{false}\,\mathbf{B}\varphi$ (φ holds always).

Note that $\varphi\mathbf{B}\psi \equiv \neg(\neg\varphi\mathbf{U}\neg\psi)$.

Example 5. Recall the ASM M in Example 3. We observe that classical model checking is a special case of model checking M. Let $\Upsilon' = \{P_1, \ldots, P_k\}$ be a vocabulary containing only set symbols, and *pebble* a nullary function symbol (as in the vocabulary of M). For every CTL* formula φ over Υ', obtain the

FBTL sentence φ' over $\Upsilon' \cup \{pebble\}$ by replacing in φ every occurrence of P_i with the atomic formula $P_i(pebble)$. Let $M_{\Upsilon'}$ be defined as M, except that the input vocabulary of $M_{\Upsilon'}$ is $\{T, 0\} \cup \Upsilon'$ and the initialization mapping of $M_{\Upsilon'}$ maps every Kripke structure $\mathcal{K}$ of form (1) to the state $(\mathcal{K}, 0^{\mathcal{K}})$. Now verify that for every CTL* formula φ over Υ' and every Kripke structure $\mathcal{K}$ of form (1),

$$\mathcal{K} \models \varphi \quad \Leftrightarrow \quad C_{M_{\Upsilon'}}(\mathcal{K}) \models \varphi'.$$

In other words, we can check whether $\mathcal{K}$ is a model of φ by deciding whether $(M_{\Upsilon'}, \varphi', \mathcal{K})$ is a positive instance of the model-checking problem for ASMs. In fact, in the next section we are going to define the model-checking problem for ASMs such that $\mathcal{K} \models \varphi$ iff $(M_{\Upsilon'}, \varphi', \mathcal{K}) \in$ MC. □

Examples of FBTL specifications motivated by concrete applications can be found in [Spi00a, Spi00b].

3 Model Checking Abstract State Machines

Recall the informal formulation of the model-checking problem for ASMs in the introduction. If the reader agrees that FBTL (or some suitable extension of this logic) is an appropriate specification language for ASMs, then the model-checking problem for ASMs can be rephrased as follows:

MC$'$: Given an ASM M, a FBTL sentence φ over the vocabulary of M, and an input $\mathcal{I}$ appropriate for M, decide whether $C_M(\mathcal{I}) \models \varphi$.

Notice, however, that this problem is still **not** a computational problem. The difficulty here is that the initialization mapping of an ASM M may not be finitely representable, in which case M cannot serve as input to a computational device.

Next, we further restrict the above problem so that it becomes a computational problem. Then, in the second part of this section, we present some results concerning the decidability and complexity of the obtained computational problem.

3.1 The Model-Checking Problem

The problem of representing the initialization mappings of ASMs in finite terms disappears if we concentrate on ASMs uniformly initialized in the following sense.

Definition 6. A class C of ASMs is *uniformly initialized* if, whenever M and M' are two ASMs in C of the same vocabulary, then M and M' have that same initialization mapping. □

With every uniformly initialized class C of ASMs and every ASM vocabulary Υ, we can associate an initialization mapping $initial^C_\Upsilon$, such that $initial^C_\Upsilon$ is the initialization mapping of every $M \in C$ of vocabulary Υ. Observe that, in the context of a uniformly initialized class C of ASMs, every $M \in C$ is uniquely determined by its vocabulary and program. More precisely, if Υ is the vocabulary and Π the program of M, then necessarily $M = (\Upsilon, initial^C_\Upsilon, \Pi)$. This motivates the following definition.

Definition 7. The *standard representation* of an ASM $(\Upsilon, initial, \Pi)$ is given by the pair (Υ, Π). □

We are now in the position to define the model-checking problem for ASMs as a computational problem. Let C be a uniformly initialized class of ASMs and F a fragment of FBTL. The *model checking problem for C-ASMs and F-specifications* is the following decision problem:

> MC(C, F): Given the standard representation of an ASM $M \in C$, a sentence $\varphi \in F$ over the vocabulary of M, and an input $\mathcal{I}$ appropriate for M, decide whether $C_M(\mathcal{I}) \models \varphi$.

Remark 8. Note the subtlety in the above definition of MC, namely that by viewing an ASM $M \in C$ as a finitely representable pair (Υ, Π) the problem of actually representing the initialization mapping of M has not been solved, but rather has been made part of the model-checking problem itself. □

Example 9. (a) For every vocabulary $\Upsilon' = \{P_1, \ldots, P_k\}$ containing only set symbols, and every CTL* formula φ over Υ', let $M_{\Upsilon'}$ and φ' be defined as in Example 5. Let C denote the class of all $M_{\Upsilon'}$, and set $F = \{\varphi' : \varphi \in \text{CTL}^*\}$. C is uniformly initialized and we have $(M_{\Upsilon'}, \varphi', \mathcal{K}) \in \text{MC}(C, F)$ iff $\mathcal{K} \models \varphi$. That is, MC$(C, F)$ coincides with the model checking problem for CTL*.

(b) One can define a class C of ASMs and a fragment F of FBTL such that MC(C, F) coincides with the *symbolic* model checking problem for CTL*. We sketch the idea. Consider a Kripke structure $\mathcal{K}$ of form (1) and a CTL* formula φ over $\{P_1, \ldots, P_k\}$. Obtain the FBTL formula φ' by replacing every occurrence of P_i with the boolean symbol p_i. (A *boolean symbol* is a nullary relation symbol.) $\mathcal{K}$ can be represented by means of boolean formulas $\gamma_T(\bar{x}, \bar{y}), \gamma_{P_1}(\bar{x}), \ldots, \gamma_{P_k}(\bar{x})$ with $\bar{x}$ and $\bar{y}$ two n-tuples of boolean variables, free$(\gamma_T) = \{\bar{x}, \bar{y}\}$, and for each P_i, free$(\gamma_{P_i}) = \{\bar{x}\}$. For simplicity, let us assume that for each P_i, $\gamma_{P_i}(\overline{false})$ evaluates to *false*. There is an ASM $M_{\mathcal{K}}$ with input vocabulary $\{0\}$, dynamic vocabulary $\{\bar{x}, \bar{p}\}$, and

```
program Π:
  choose y_1, ..., y_n ∈ {true, false} : γ_T(x̄, ȳ)
     x̄ := ȳ
     if γ_P1(ȳ) then p_1 := true else p_1 := false
     ...
     if γ_Pk(ȳ) then p_k := true else p_k := false
```

such that for any input $\mathcal{I}$ appropriate for $M_{\mathcal{K}}$, $\mathcal{K} \models \varphi \Leftrightarrow C_{M_{\mathcal{K}}}(\mathcal{I}) \models \varphi'$. □

In the remainder of this section we investigate the decidability and complexity of MC(C, F).

3.2 Decidability and Complexity

We first observe that the model-checking problem is decidable for uniformly initialized ASMs whose initial states are finite and computable from the inputs.

Lemma 10. *Let C be a uniformly initialized class of ASMs such that there exists an algorithm A_C as follows. For every ASM vocabulary Υ and every input $\mathcal{I}$ over Υ, A_C on input $(\Upsilon, \mathcal{I})$ outputs $initial^C_\Upsilon(\mathcal{I})$ (where $initial^C_\Upsilon$ is defined as below Definition 6). Then* $\textsc{mc}(C, \mathrm{FBTL})$ *is decidable.*

Proof. Given an instance $(M, \varphi, \mathcal{I})$ of $\textsc{mc}(C, \mathrm{FBTL})$, where the vocabulary of M is, say, Υ, we can run A_C on $(\Upsilon, \mathcal{I})$ to obtain the initial state of M on $\mathcal{I}$. Since this state is finite, so is $C_M(\mathcal{I})$. Hence, we can construct $C_M(\mathcal{I})$ explicitly and check whether $C_M(\mathcal{I}) \models \varphi$ holds. $\square$

Applications of this lemma can be found in [Win97, CW99, Win00]. Note however that the decision procedure outlined in the proof of the lemma is often too expensive to be useful in practice, even if the complexity of the algorithm A_C is low. For example, the number of reachable states of an ASM M employing a non-nullary dynamic relation in general grows exponentially in the size of the initial state. Consequently, the space required for constructing a computation graph of M explicitly can be exponential in the size of the input, independent of A_C.

Next, we present a class C of ASMs and a fragment F of FBTL for which a useful restriction of $\textsc{mc}(C, F)$ is in PSPACE.

Definition 11. An ASM $M = (\Upsilon, initial, \Pi)$ is *finitely initialized* if for every input $\mathcal{I}$ appropriate for M,

1. the universe of $initial(\mathcal{I})$ is that of $\mathcal{I}$, and
2. for every relation symbol $R \in \Upsilon - \Upsilon_{\mathrm{in}}$, every k-ary function symbol $f \in \Upsilon - \Upsilon_{\mathrm{in}}$, and every k-tuple $\bar{a}$ of elements in $initial(\mathcal{I})$,

$$R^{initial(\mathcal{I})} = \varnothing \quad \text{and} \quad f^{initial(\mathcal{I})}(\bar{a}) = 0.$$

$\square$

Observe that the initialization mapping of every finitely initialized ASM M is uniquely determined by the vocabulary of M. In particular, every class of finitely initialized ASMs is uniformly initialized and for every such class C there exists an algorithm A_C as in Lemma 10.

Remark 12. (a) On the first glance, the second condition in Definition 11 may seem to render finitely initialized ASMs useless for practical purposes, because in applications the initial interpretations of the symbols in $\Upsilon - \Upsilon_{\mathrm{in}}$ must often satisfy certain conditions. In particular, it may not be desirable to have all symbols in $\Upsilon - \Upsilon_{\mathrm{in}}$ be initialized as in the second condition of the definition. But note that the initial interpretations of these symbols can be seen as part of the input of an ASM. For example, consider a finitely initialized ASM M whose nullary dynamic function *pebble* should assume, instead of 0, the value 42 in the initial state. This can be achieved by modifying M as follows. Let Π be the program of M, *pebbleInitial* a new constant symbol, and *running* a new boolean symbol. Replace Π with the following program:

```
program Π':
  if ¬running then
    pebble := pebbleInitial
    running := true
  if running then
    Π
```

Now, while model checking M, consider only input structures in which the input symbol *pebbleInitial* is interpreted as 42.

(b) Suppose that the second condition in Definition 11 is relaxed as follows: there is a distinguished symbol $S \in \Upsilon - \Upsilon_{\text{in}}$ whose initial interpretation depends on Υ but is not restricted otherwise. We display a class C of simple ASMs finitely initialized in the relaxed sense and a simple fragment F of FBTL such that $\textsc{mc}(C, F)$ is undecidable.

W.l.o.g., we may assume that S is a boolean symbol and that every ASM vocabulary contains S. Choose some undecidable problem

$$P \subseteq \{\Upsilon : \Upsilon \text{ is an ASM vocabulary}\}.$$

(For instance, let TM_Υ be the Turing machine whose encoding—in some fixed standard binary encoding—equals the binary representation of the number of symbols in Υ. Then $P := \{\Upsilon : TM_\Upsilon \text{ halts on the empty word}\}$ is undecidable, as a reduction of the halting problem for Turing machines shows.) For every ASM vocabulary Υ, let $M_\Upsilon := (\mathtt{skip}, \mathit{initial}_\Upsilon, \Upsilon)$ be a finitely initialized ASM, except that now for every input $\mathcal{I}$ appropriate for M, $\mathit{initial}_\Upsilon(\mathcal{I}) \models S$ iff $\Upsilon \in P$. Let C denote the class of all M_Υ, and set $F = \{S\}$. For any input $\mathcal{I}$ appropriate for M, $\Upsilon \mapsto (M_\Upsilon, S, \mathcal{I})$ is a reduction of P to $\textsc{mc}(C, F)$. This implies that $\textsc{mc}(C, F)$ is undecidable. □

In favor of a succinct formulation of the next theorem we introduce some additional notation.

Definition 13. Let T denote the closure of the set of first-order formulas under the rules for negation and disjunction, and the following rule:

(T) If φ and ψ are formulas, then $\mathbf{X}\varphi$, $\varphi\mathbf{U}\psi$, and $\varphi\mathbf{B}\psi$ are formulas.

The *universal closure of* T, denoted UT, is the set of formulas of the form $\mathbf{A}\forall\bar{x}\varphi$ with $\varphi \in \mathrm{T}$ and $\mathrm{free}(\varphi) \subseteq \{\bar{x}\}$, where $\mathrm{free}(\varphi)$ is defined in the obvious way. □

We define a *restriction* of the model-checking problem for ASMs. An investigation of this restriction is motivated by the observation that the arities of relations and functions used in practice tend to be rather small. Indeed, for practical purposes it often suffices to solve $\textsc{mc}$ for ASMs whose vocabulary contains only symbols of arity $\leq m$, for some a priori fixed natural number m. Let $\textsc{mc}_m(C, F)$ denote the restriction of $\textsc{mc}(C, F)$ to instances where only symbols of arity $\leq m$ occur.

The following theorem generalizes a result in [Spi00a]. Note that UT can be viewed as a fragment of FBTL.

Theorem 14. *Let C be the class of finitely initialized ASMs whose program does not contain* `import` *or* `choose`*. For any natural number m, $\textsc{mc}_m(C, \mathrm{UT})$ is* Pspace*-complete.*

We already encounter Pspace-hardness of $\textsc{mc}_m(C, \mathrm{UT})$ if $m = 0$ and C is the class of finitely initialized ASMs of the form $(\texttt{skip}, \mathit{initial}_\Upsilon, \Upsilon)$. This follows by a reduction of the Pspace-complete model-checking problem for propositional temporal logic [SC85, Eme90].

As already pointed out in the introduction, a decision procedure for $\textsc{mc}(C, F)$ can be useful for the verification of ASMs that are supposed to run correctly on a small number of inputs only. However, for applications where correctness of an ASM has to be ensured for a large number of inputs—or even infinitely many inputs—a solution of $\textsc{mc}(C, F)$ does not suffice. For such applications we have to solve the model-checking problem for *all* admissible inputs. This will be our main concern in the next section.

4 Verifying Abstract State Machines

We study the problem of model checking ASMs against FBTL specifications for *all* inputs. After formally defining this problem, which we call the *verification problem* for ASMs, we provide some evidence that it subsumes the model-checking problem for ASMs and investigate its decidability and complexity.

4.1 The Verification Problem

Let C be a uniformly initialized class of ASMs and F a fragment of FBTL. *Verifying C-ASMs against F-specifications* means solving the following decision problem:

$\textsc{verify}(C, F)$: Given the standard representation of an ASM $M \in C$ and a sentence $\varphi \in F$ over the vocabulary of M, decide whether for every input $\mathcal{I}$ appropriate for M, $C_M(\mathcal{I}) \models \varphi$.

Remark 15. One may object that the above formulation of $\textsc{verify}$ does not adequately reflect real-life verification of ASMs, as in applications one is often interested in verifying an ASM only for inputs that satisfy certain conditions. Note however that these conditions can be viewed as part of a specification. For example, suppose that the FBTL sentence ψ describes all admissible inputs of an ASM M. That is, an input $\mathcal{I}$ is considered to be admissible for M iff $\mathcal{I} \models \psi$. Then $(M, \psi \to \varphi)$ is a positive instance of $\textsc{verify}$ iff for every admissible input $\mathcal{I}$ appropriate for M, $C_M(\mathcal{I}) \models \varphi$. □

Not surprising, $\textsc{verify}(C, F)$ is in general undecidable, even for classes C of simple ASMs and simple fragments F of FBTL. For instance, recall the reduction of the undecidable problem P to $\textsc{mc}(C, F)$ in Remark 12 (b) and observe that the same reduction (with $\mathcal{I}$ now removed from the image of every Υ) also reduces P to $\textsc{verify}(C, F)$.

Proviso 16. *In the remainder of this section we restrict ourselves to finitely initialized ASMs (see Definition 11).* □

4.2 Model Checking vs Verification

We show that $\textsc{mc}(C, F)$ is polynomial-time reducible to $\textsc{verify}(C, F)$ if C and F satisfy certain closure conditions. In favor of a concise presentation of the result we introduce a restricted type of first-order quantification, which we call *witness-bounded quantification*. In the following, FO denotes first-order logic (with equality).

Witness-bounded quantification. A *witness set* W is a finite set of variables and constant symbols. For a witness set W and a variable x not in W, we write $(x \in W)$ instead of $(\bigvee_{\nu \in W} x = \nu)$. Intuitively, $(x \in W)$ holds iff the interpretation of x matches the interpretation of some symbol in W.

Definition 17. The *witness-bound fragment* of FO, denoted FO^{W}, is obtained from FO by replacing the rule for (unrestricted) quantification with the following rule for *witness-bounded quantification*:

(WBQ) If W is a witness set, x a variable not in W, and φ a formula, then $(\exists x \in W)\varphi$ and $(\forall x \in W)\varphi$ are formulas.

The free and bound variables of FO^{W} formulas are defined as usual. In particular, x occurs bound in $(\exists x \in W)\varphi$ and $(\forall x \in W)\varphi$, while all variables in the witness set W occur free. □

We view FO^{W} as a fragment of FO where formulas of the form $(\exists x \in W)\varphi$ and $(\forall x \in W)\varphi$ are mere abbreviations for $\exists x(x \in W \wedge \varphi)$ and $\forall x(x \in W \rightarrow \varphi)$, respectively.

Reduction of $\textsc{mc}$ **to** $\textsc{verify}$**.** Proceeding toward a reduction of $\textsc{mc}$ to $\textsc{verify}$, we next define witness-bounded versions of ASMs and FBTL formulas. Let W be a witness set containing only constant symbols. For every FBTL formula φ, obtain φ^W from φ by replacing every first-order quantifier with the corresponding W-bounded quantifier (e.g., replace $\exists x$ with $(\exists x \in W)$, and $\forall x$ with $(\forall x \in W)$). For every ASM $M = (\Upsilon, \mathit{initial}, \Pi)$ with $W \subseteq \Upsilon_{\mathrm{in}}$, let $M^W := (\Upsilon, \mathit{initial}, \Pi^W)$ where Π^W is obtain by induction on the construction of Π:

- If Π is `skip` or an update, then $\Pi^W := \Pi$.
- If $\Pi = (\texttt{if } \varphi \texttt{ then } \Pi_0)$, then $\Pi^W := (\texttt{if } \varphi^W \texttt{ then } \Pi_0^W)$.
- If $\Pi = (\Pi_0 || \Pi_1)$, then $\Pi^W := (\Pi_0^W || \Pi_1^W)$.
- If $\Pi = (\texttt{do-forall } x : \varphi, \Pi_0)$, then $\Pi^W := (\texttt{do-forall } x \in W : \varphi^W, \Pi_0^W)$.

If Π is a `choose` or `import` rule, proceed similar to the last case. For a class C of ASMs and a fragment F of FBTL, let C^{W} and F^{W} be defined by:

$$\begin{aligned} C^{\mathrm{W}} &:= \{M^W : M = (\Upsilon, \mathit{initial}, \Pi) \in C, \\ &\qquad W \subseteq \Upsilon_{\mathrm{in}} \text{ is a set of constant symbols}\} \\ F^{\mathrm{W}} &:= \{\varphi^W : \varphi \in F,\ W \text{ is a set of constant symbols}\}. \end{aligned}$$

Lemma 18. *Let C_0 be the class of finitely initialized ASMs whose program does not contain* `import` *and whose input (resp. external) vocabulary does not contain function symbols of arity ≥ 1 (resp. ≥ 0). Furthermore, let F be a fragment of* FBTL *containing for every boolean symbol b the formula* $\mathbf{EX}b$*. For any class $C' \supseteq C_0^{\mathrm{W}}$ and any fragment $F' \supseteq F^{\mathrm{W}}$ closed under the boolean operators,* $\textsc{mc}(C_0, F)$ *is polynomial-time reducible to* $\textsc{verify}(C', F')$.

Proof. (Sketch.) Consider an instance $(M, \varphi, \mathcal{I})$ of $\textsc{mc}(C, F)$ with $M = (\Upsilon, \mathit{initial}_\Upsilon, \Pi)$. W.l.o.g., we may assume that every element in $\mathcal{I}$ is denoted by some constant symbol in Υ_{in}. (If this is not the case, enrich Υ_{in} with new constant symbols and add interpretations of the new symbols to $\mathcal{I}$. This modification of $\mathcal{I}$ is clearly polynomial-time computable.) We construct an instance (M', φ') of $\textsc{verify}(C^{\mathrm{W}}, F')$ which is positive iff $(M, \varphi, \mathcal{I})$ is a positive instance of $\textsc{mc}(C, F)$.

Let $\chi_\mathcal{I} := \bigwedge_{\mathcal{I} \models \gamma} \gamma$ where γ ranges in the set of atomic and negated atomic sentences over Υ_{in}. $\chi_\mathcal{I}$ is a quantifier-free FO sentence over Υ_{in} and can be constructed from $\mathcal{I}$ in polynomial time. For every finite structure $\mathcal{I}'$ over Υ_{in} we have $\mathcal{I}' \models \chi_\mathcal{I}$ iff there exists a substructure of $\mathcal{I}'$ isomorphic to $\mathcal{I}$.

To the definition of M' and φ'. Let W be the set of constant symbols in Υ_{in}, and *error* a boolean symbol not in Υ. Set $\varphi' = (\neg \mathbf{EX}\mathit{error} \to \varphi^W)$ and $M' = (\Upsilon, \mathit{initial}_\Upsilon, \Pi')$ with $\Pi' := (\Pi^W \,||\, \texttt{if}\ \neg\chi_\mathcal{I}\ \texttt{then}\ \mathit{error})$. (M', φ') can be obtained from $(M, \varphi, \mathcal{I})$ in polynomial time and is an instance of $\textsc{verify}(C^{\mathrm{W}}, F')$. □

Let C_0 be as in the above lemma. The proof of the lemma shows that, if $C \subseteq C_0$ and $F \subseteq$ FBTL have reasonable closure properties and $C^{\mathrm{W}} \subseteq C$ and $F^{\mathrm{W}} \subseteq F$, then $\textsc{mc}(C, F)$ is polynomial-time reducible to $\textsc{verify}(C, F)$.

4.3 Decidability and Complexity

The next proposition provides a sufficient condition for the decidability of the verification problem. Let us first recall two decision problems from logic.

Finite satisfiability and finite validity. Let L be a logic, Υ a vocabulary, and φ an L-sentence over Υ. φ is called *finitely satisfiable* if there exists a finite structure $\mathcal{A}$ over Υ with $\mathcal{A} \models \varphi$. φ is called *finitely valid* if for every finite structure $\mathcal{A}$ over Υ, $\mathcal{A} \models \varphi$. By $\textsc{fin-sat}(L)$ (resp. $\textsc{fin-val}(L)$) we denote the problem of deciding finite satisfiability (resp. finite validity) of a given L-sentence.

Proposition 19. *(a) Let C be a class of ASMs and F a fragment of* FBTL*. If there exists a logic L satisfying the following two conditions, then* $\textsc{verify}(C, F)$ *is decidable.*

1. $\textsc{fin-val}(L)$ *is decidable.*
2. *There exists a computable function which maps every instance (M, φ) of* $\textsc{verify}(C, F)$ *to an L-sentence $\chi_{M,\varphi}$ over the input vocabulary of M, such that for every input $\mathcal{I}$ appropriate for M,*

$$C_M(\mathcal{I}) \models \varphi \quad \Leftrightarrow \quad \mathcal{I} \models \chi_{M,\varphi}.$$

(b) Let $\neg F$ denote the set of negated F-sentences. VERIFY$(C, \neg F)$ *is decidable if in the first condition instead of* FIN-VAL(L), FIN-SAT(L) *is decidable.*

Proof. To assertion (a). The second condition immediately implies that $(M, \varphi) \mapsto \chi_{M,\varphi}$ is a reduction of VERIFY(C, F) to FIN-VAL(L). The latter problem is decidable by assumption. To assertion (b). Verify the following chain of equivalences: $(M, \neg\varphi) \in$ VERIFY$(C, \neg F)$ iff $C_M(\mathcal{I}) \models \neg\varphi$ for every $\mathcal{I}$ (appropriate for M) iff $C_M(\mathcal{I}) \not\models \varphi$ for every $\mathcal{I}$ iff $\mathcal{I} \not\models \chi_{M,\varphi}$ for every $\mathcal{I}$ iff $\chi_{M,\varphi} \notin$ FIN-SAT(L). Hence, $(M, \neg\varphi) \mapsto \chi_{M,\varphi}$ is a reduction of VERIFY$(C, \neg F)$ to the complement of FIN-SAT(L). Decidability of FIN-SAT(L) then implies assertion (b). □

Of course, the main challenge in applying Proposition 19 is to find a logic L that satisfies the two conditions in the proposition. As possible candidates for L we propose the following logics:

- existential transitive closure logic (E+TC),
- existential least fixed-point logic (E+LFP) [BG87, EF95], and
- SO($\exists$), i.e., the set of second-order formulas in prenex normal form, whose quantifier prefix is an arbitrary string of SO quantifiers followed be a sting of existential FO quantifiers [Ros99].

Both FIN-VAL and FIN-SAT are decidable for each of these logics if one restricts attention to formulas over relational vocabularies [LMSS93, Ros99].

In the remainder of this section we recall some results concerning the decidability of VERIFY from [Spi99]. The main positive result there was obtained by means of an application of Proposition 19.

Definition 20. Let Υ be an ASM vocabulary where Υ_{dyn} and Υ_{ext} contain only nullary symbols. *Sequential nullary programs* over Υ are defined inductively:

- **Updates:** For every relation symbol $b \in \Upsilon_{\text{dyn}}$, every function symbol $v \in \Upsilon_{\text{dyn}}$, and every term t over Υ, each of the following is a sequential nullary program: $b := \mathit{true}$, $b := \mathit{false}$, $v := t$.
- **Conditionals:** If Π is a sequential nullary program and φ a witness-bounded formula over Υ, then $(\texttt{if}\ \varphi\ \texttt{then}\ \Pi)$ is a sequential nullary program.
- **Parallel composition:** If Π_0 and Π_1 are sequential nullary programs, then $(\Pi_0 || \Pi_1)$ is a sequential nullary program.
- **Non-deterministic choice:** If Π is a sequential nullary program, $\bar{x}$ a tuple of pairwise distinct variables, and φ a witness-bounded formula over Υ such that $\exists \bar{x} \varphi$ is finitely valid, then $(\texttt{choose}\ \bar{x} : \varphi,\ \Pi)$ is a sequential nullary program.

The *free* and *bound variables* of a sequential nullary program are defined in the obvious way.

A *sequential nullary ASM* is a finitely initialized ASM whose program is a sequential nullary program without free variables. SN-ASM$_{\text{rel}}$ (resp. SN-ASM$_{\text{fct}}$) denotes the class of sequential nullary ASMs whose input vocabulary contains only relation and constant symbols (resp. contains at least two non-nullary symbols, one of which is a function symbol). □

The next definition introduces two fragments of FBTL, denoted ETE and UTU. Informally speaking, the formulas in ETE (resp. UTU) are built from atomic formulas by means of disjunction, conjunction, existential (resp. universal) quantification (applicable only to state formulas), the temporal operators $\mathbf{X}$, $\mathbf{U}$, and $\mathbf{B}$, and the path quantifier $\mathbf{E}$ (resp. $\mathbf{A}$).

Definition 21. ETE denotes the set of FBTL formulas derivable by means of rules **(S1)**, **(S2)**, **(P1)**, **(P2)** in Definition 4, and the following two formula-formation rules:

(SP1)' If φ and ψ are state (resp. path) formulas, then $\varphi \vee \psi$ and $\varphi \wedge \psi$ are state (resp. path) formulas.

(SP2)' If x is a variable and φ a state formula, then $\exists x \varphi$ is a state formula.

UTU denotes the set of negated ETE formulas. □

Let $\textsc{verify}_m(C, F)$ denote the restriction of $\textsc{verify}(C, F)$ to instances where only symbols of arity $\leq m$ occur. Again, solving $\textsc{verify}_m(C, F)$ instead of $\textsc{verify}(C, F)$ should be no serious obstacle for practical purposes.

Theorem 22 **([Spi99]).** *For any natural number m, the problem*

$$\textsc{verify}_m(\text{SN-ASM}_{\text{rel}}, \text{ETE} \cup \text{UTU})$$

is PSPACE*-complete. In other words, verifying sequential nullary ASMs with relational input against* (ETE ∪ UTU)*-specifications is a* PSPACE*-complete problem, given that the maximal arity of the employed input relations is a priori bounded.*

The proof of the theorem closely follows a construction due to Immerman and Vardi that was first presented in [IV97] as a translation of CTL* into (FO+TC). Using this construction one can show that for $C = \text{SN-ASM}_{\text{rel}}$, $F = \text{ETE}$, and $L = (\text{E+TC})$ the second condition in Proposition 19 is satisfied. The containment assertion of the theorem is then implied by the following two observations:

1. Finite validity and finite satisfiability of (E+TC) sentences over relational vocabularies is *decidable in* PSPACE if one imposes an upper bound m on the arities of the occurring relation symbols [Spi99, Spi00b].
2. There exists a *polynomial-time* computable function as in the second condition of Proposition 19.

$\textsc{verify}_m(C, F)$ is already PSPACE-hard if $m = 0$, F contains $\mathbf{EX}\mathit{accept}$, and C includes all ASMs in $\text{SN-ASM}_{\text{rel}}$ whose external vocabulary is empty and whose dynamic vocabulary contains only boolean symbols. This follows by a reduction of the PSPACE-complete satisfiability problem for quantified boolean formulas.

Unfortunately, Theorem 22 does not hold for sequential nullary ASMs whose inputs contain functions, as the following theorem shows. For the sake of brevity we write $\textsc{liveness}(C)$ instead of $\textsc{verify}(C, \{\mathbf{EF}\mathit{accept}\})$ and $\textsc{safety}(C)$ instead of $\textsc{verify}(C, \{\mathbf{AG}\neg \mathit{error}\})$.

Theorem 23 ([Spi99]). *The two problems* LIVENESS(SN-ASM$_{\mathrm{fct}}$) *and* SAFETY (SN-ASM$_{\mathrm{fct}}$) *are both undecidable. Moreover,* SAFETY(C) *is undecidable for every class C that includes all deterministic ASMs in* SN-ASM$_{\mathrm{fct}}$ *whose external vocabulary is empty and whose dynamic vocabulary consists of two nullary function symbols and arbitrarily many nullary relation symbols.*

The proof of this theorem is by reduction of the halting problem for Turing machines. It is based on the following two observations:

1. Inputs that contain non-nullary functions suffice to encode bit-strings.
2. Two nullary dynamic functions suffice to check whether a bit-string encodes an accepting computation of a Turing machine.

An analysis of the proof indicates that automatic verification of even very simple ASMs whose inputs can be used for encoding bit-strings is not feasible. Nevertheless, Theorem 22 and results in [Spi00a] show that restricted variants of ASMs with relational inputs can be verified automatically.

Acknowledgements. Thanks to the anonymous referees.

References

[BG87] A. Blass and Y. Gurevich. Existential fixed-point logic. In E. Börger, editor, *Computation Theory and Logic*, volume 270 of *Lecture Notes in Computer Science*, pages 20–36. Springer-Verlag, 1987.

[BG97] A. Blass and Y. Gurevich. The Linear Time Hierarchy Theorems for Abstract State Machines. *Journal of Universal Computer Science*, 3(4):247–278, 1997.

[BGdB00] A. Blass, Y. Gurevich, and J. Van den Bussche. Abstract State Machines and Computationally Complete Query Languages. This volume.

[BGS99] A. Blass, Y. Gurevich, and S. Shelah. Choiceless Polynomial Time. Technical Report MSR-TR-99-08, Mircosoft Research, 1999.

[BH98] E. Börger and J. Huggins. Abstract State Machines 1988–1998: Commented ASM Bibliography. *Bulletin of the EATCS*, 64:105–127, February 1998.

[BS00] D. Beauquier and A. Slissenko. Verification of Timed Algorithms: Gurevich Abstract State Machines versus First Order Timed Logic. Technical Report TIK-Report 87, ETH Zürich, March 2000.

[Bör95] E. Börger. Why Use Evolving Algebras for Hardware and Software Engineering? In M. Bartosek, J. Staudek, and J. Wiederman, editors, *Proceedings of 22nd Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM '95)*, volume 1012 of *Lecture Notes in Computer Science*, pages 236–271. Springer Verlag, 1995.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic. *ACM Trans. on Prog. Lang. and Sys.*, 8(2):244–263, April 1986.

[CW99] G. Del Castillo and K. Winter. Model Checking Support for the ASM High-Level Language. Technical Report TR-RI-99-209, Universität-GH Paderborn, June 1999.

[EF95] H. D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[Eme90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–11072. Elsevier Science Publishers B.V., 1990.

[GR00] A. Gargantini and E. Riccobene. Encoding Abstract State Machines in PVS. This volume.

[GS99] E. Grädel and M. Spielmann. Logspace Reducibility via Abstract State Machines. In J. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods (FM '99)*, volume 1709 of *Lecture Notes in Computer Science*, pages 1738–1757. Springer-Verlag, 1999.

[Gur95] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[Gur97] Y. Gurevich. May 1997 Draft of the ASM Guide. Technical Report CSE-TR-336-97, University of Michigan, May 1997.

[Gur99] Y. Gurevich. The Sequential ASM Thesis. *Bulletin of the EATCS*, 67:93–124, 1999.

[IV97] N. Immerman and M.Y. Vardi. Model Checking and Transitive Closure Logic. In *Proceedings of 9th International Conference on Computer-Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 291–302. Springer-Verlag, 1997.

[LMSS93] A. Levy, I. Mumick, Y. Sagiv, and O. Shmueli. Equivalence, Query-Reachability, and Satisfiability in Datalog Extensions. In *Proceedings of 12th ACM Symposium on Principles of Database Systems (PODS '93)*, pages 109–122, 1993.

[Ros99] E. Rosen. An existential fragment of second order logic. *Archive for Mathematical Logic*, 38:217–234, 1999.

[SC85] A.P. Sistla and E.M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the Association for Computing Machinery*, 32(3):733–749, July 1985.

[Spi99] M. Spielmann. Automatic Verification of Abstract State Machines. In *Proceedings of 11th International Conference on Computer-Aided Verification (CAV '99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 431–442. Springer-Verlag, 1999.

[Spi00a] M. Spielmann. Verification of Relational Transducers for Electronic Commerce. In *Proceedings of 19th ACM Symposium on Principles of Database Systems (PODS 2000)*. ACM Press, 2000. To appear.

[Spi00b] M. Spielmann. *Abstract State Machines: Verification Problems and Complexity*. PhD thesis, RWTH Aachen. In preparation.

[Web] ASM Web Site. http://www.eecs.umich.edu/gasm. Maintained by J. Huggins.

[Win97] K. Winter. Model Checking for Abstract State Machines. *Journal of Universal Computer Science*, 3(5):689–701, 1997.

[Win00] K. Winter. Methodology for Model Checking ASM: Lessons learned from the FLASH Case Study. This volume.