

A Logic for Abstract State Machines

Robert F. Stärk and Stanislas Nanchen

Computer Science Department, ETH Zürich, CH-8092 Zürich, Switzerland
staerk@inf.ethz.ch and nanchen@inf.ethz.ch

Abstract. We introduce a logic for sequential, non distributed Abstract State Machines. Unlike other logics for ASMs which are based on dynamic logic, our logic is based on atomic propositions for the function updates of transition rules. We do not assume that the transition rules of ASMs are in normal form, for example, that they concern distinct cases. Instead we allow structuring concepts of ASM rules including sequential composition and possibly recursive submachine calls. We show that several axioms that have been proposed for reasoning about ASMs are derivable in our system and that the logic is complete for hierarchical (non-recursive) ASMs.

Keywords: Abstract State Machines, dynamic logic, modal logic, logical foundations of specification languages.

1 Introduction

Gurevich's Abstract State Machines (ASMs) [5] are widely used for the specification of software, hardware, algorithms, and the semantics of programming languages [1]. Most logics that have been proposed for ASMs are based on variants of dynamic logic. There are, however, fundamental differences between the imperative programs of dynamic logic and ASMs. In dynamic logic, states are represented with variables. In ASMs, states are represented with dynamic functions. The fundamental program constructs in dynamic logic are non-deterministic iteration (star operator) and sequential composition. The basic transition rules of ASMs consist of parallel function updates. Since parallel function updates may conflict, a logic for ASMs must have a clear notion of consistency of transition rules. Therefore, rather than to encode ASMs into imperative programs of dynamic logic or to extend the axioms and rules of dynamic logic, we propose new axioms and rules which are directly based on an update predicate for transition rules. Since ASMs are special instances of transition systems, our logic contains a modal operator, too [14].

What comes closest to our system is known as dynamic logic with array assignments [6,7]. The substitution principle which is used in its axiomatization is derivable in our system (Lemma 9). The dynamic logic with array assignments,

Appeared in: L. Fribourg, editor, *Computer Science Logic CSL '01*, Paris, Springer-Verlag, Lecture Notes in Computer Science 2142, pages 217-231, 2001.

© Springer-Verlag, see <http://www.springer.de/comp/lncs/>

however, is not concerned with parallel execution of assignments and therefore does not need a notion of consistency.

Groenboom and Renardel de Lavalette introduce in [4] the *Formal Language for Evolving Algebras* (FLEA), a system for formal reasoning about abstract state machines. Their system is in the tradition of dynamic logic and contains for every rule R a modal operator $[R]\varphi$ with the intended meaning that φ holds always after the execution of R . The logic of their formal language contains besides *true* and *false* a third truth-value which stands for *undefined*. Although they consider parallel composition of transition rules, they have no formal notion of consistency in their system. We adopt their modal operator $[R]\varphi$ such that the basic axioms of their system are derivable in our logic (Lemma 5). We use, however, the two-valued logic of the classical predicate logic.

Our system is designed to deal with Börger and Schmid's *named parameterized ASM rules* [2] which include also recursive definitions of transition rules. Recursive rule definitions in combination with sequential compositions of transition rules are maybe too powerful and not in the spirit of the basic ASM concept [5]. Nevertheless we include them in our logic and solve the technical problems that arise with an explicit definedness predicate.

Schönege extends in [11] the dynamic logic of the KIV system (Karlsruhe Interactive Verifier) to turn it into a tool for reasoning about abstract state machines. The transition rules of ASMs are considered as imperative programs of dynamic logic. While-programs of dynamic logic are used as an interpreter for abstract state machines; loop-programs are used to apply an ASM a finite number of times. Schönege's rules for simultaneous function updates and parallel composition of transition rules are derivable in our system (Lemma 7). His sequent calculus is mainly designed as a practical extension of the KIV system and not as a foundation for reasoning about transition rules and ASMs.

Schellhorn and Ahrendt simulate in [10] abstract state machines in the KIV system by formalizing dynamic functions as association lists, serializing parallel updates and transforming transition rules into flat imperative program of the underlying dynamic logic. Schellhorn is able to fully mechanize a correctness proof of the Prolog to WAM compilation in his dissertation in [9]. He argues that the inconsistency of an ASM (clash in simultaneous updates) can only be detected when the ASM is in normal form and that the transformation of the ASM into normal form by a pre-processor is more efficient than a formalization of consistency in terms of logical axioms as we do it in our system. We do think that a suitable theorem prover can automatically process and simplify our consistency conditions (Lemma 3 and Table 5) without problems.

Poetzsch-Heffter introduces in [8] a basic logic for a class of ASMs consisting of simultaneous updates of 0-ary functions (dynamic constants) and if-then-else rules only. His basic axiom states that the truth of the *weakest backwards transformer* of a formula implies the truth of the formula in the next state. He then derives partial correctness logics for a class of simple imperative programming languages by specifying their semantics with ASMs of his restricted class. His basic axiom is derivable in our system (Lemma 10).

Gargantini and Riccobene show in [3] how the PVS theorem prover can provide tool support for ASMs. They show how ASMs can be encoded in PVS (and hence in the underlying formal system which is Church’s simple theory of types). Functions are encoded as PVS functions and an interpreter for ASMs is implemented in PVS. The parallel rule application is serialized. The abstraction level of the abstract states is preserved by assuming properties of certain static functions rather than by implementing (explicitly defining) them in PVS. They do not provide a technique for proving consistency of ASMs as we do in our logic. We think that for verification of large ASMs a theorem prover like PVS should directly provide support for transition rules such that the overhead introduced by the encoding is avoided (cf. [12]).

The plan of this paper is as follows. In Sect. 2 we give a short overview on ASMs with sequential composition and (possibly recursive) rule definitions. After some considerations on formalizing the consistency of transition rules in Sect. 3, we introduce in Sect. 4 the basic axioms and rules of our logic and show that several useful principles are derivable. In Sect. 5 we prove that the logic is complete for hierarchical (non-recursive) ASMs.

2 ASM Rules and Update Sets

The notion of an *abstract state* is the classical notion of a mathematical structure \mathfrak{A} for a vocabulary Σ consisting of a non-empty set $|\mathfrak{A}|$ and of functions $f^{\mathfrak{A}}$ from $|\mathfrak{A}|^n$ to $|\mathfrak{A}|$ for each n -ary function name f of Σ . The terms s, t and the first-order formulas φ, ψ of the vocabulary Σ are interpreted as usual in the structure \mathfrak{A} with respect to a variable assignment ζ . The value of a term t in the structure \mathfrak{A} under ζ is denoted by $\llbracket t \rrbracket_{\zeta}^{\mathfrak{A}}$; the truth value of a formula φ in \mathfrak{A} under ζ is denoted by $\llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}}$ (see Table 1). The variable assignment which is obtained from ζ by assigning the element a to the variable x is denoted by $\zeta \frac{a}{x}$.

Abstract State Machines (ASMs) are systems of finitely many *transition rules* which update some of the functions of the vocabulary Σ in a given state at some arguments. The functions of the vocabulary Σ are divided into *static* functions which cannot be updated by an ASM and *dynamic* ones which typically do change as a consequence of updates by the ASM. The transition rules R, S of an ASM are syntactic expressions generated as follows (the function arguments can be read as vectors):

1. *Skip Rule:* **skip**
Meaning: Do nothing.
2. *Update Rule:* $f(t) := s$
Syntactic condition: f is a dynamic function name of Σ
Meaning: In the next state, the value of f at the argument t is updated to s .
3. *Block Rule:* $R \ S$
Meaning: R and S are executed in parallel.
4. *Conditional Rule:* **if φ then R else S**
Meaning: If φ is true, then execute R , otherwise execute S .

5. *Let Rule:* $\text{let } x = t \text{ in } R$
Meaning: Assign the value of t to x and execute R .
6. *Forall Rule:* $\text{forall } x \text{ with } \varphi \text{ do } R$
Meaning: Execute R in parallel for each x satisfying φ .
7. *Sequence Rule:* $R; S$
Meaning: R and S are executed sequentially, first R and then S .
8. *Call Rule:* $\rho(t)$
Meaning: Call ρ with parameter t .

A *rule definition* for a rule name ρ is an expression $\rho(x) = R$, where R is a transition rule in which there are no free occurrences of variables except of x . The calling convention is *lazy*. This means that in a call $\rho(t)$ the variable x is replaced in the body R of the rule by the parameter t . The parameter t is not evaluated in the state where the rule is called but only later when it is used in the body (maybe in different states due to sequential compositions). Call-by-value evaluation of rule calls can be simulated as follows:

$$\rho(y) = \text{let } x = y \text{ in } R$$

Then upon calling $\rho(t)$ the parameter t is evaluated in the same state.

Definition 1 (ASM). An *abstract state machine* M consists of a vocabulary Σ , an initial state \mathfrak{A} for Σ , a rule definition for each rule name, and a distinguished rule name of arity zero called the *main rule name* of the machine.

The semantics of transition rules is given by sets of updates. Since due to the parallelism (in the Block and the Forall rules), a transition rule may prescribe to update the same function at the same arguments several times, such updates are required to be consistent. The concept of consistent update sets is made more precise by the following definitions.

Definition 2 (Update). An *update* for \mathfrak{A} is a triple $\langle f, a, b \rangle$, where f is a dynamic function name, and a and b are elements of $|\mathfrak{A}|$.

The meaning of the update is that the interpretation of the function f in \mathfrak{A} has to be changed at the argument a to the value b . The pair of the first two components of an update is called a *location*. An update specifies how the function table of a dynamic function has to be updated at the corresponding location. An *update set* is a set of updates.

Definition 3 (Consistent update set). An update set U is called *consistent*, if it satisfies the following property: If $\langle f, a, b \rangle \in U$ and $\langle f, a, c \rangle \in U$, then $b = c$.

This means that a consistent update set contains for each function and each argument at most one value. If an update set U is consistent, it can be fired in a given state. The result is a new state in which the interpretations of dynamic function names are changed according to U . The interpretations of static function names are the same as in the old state.

$\llbracket s = t \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if } \llbracket s \rrbracket_{\zeta}^{\mathfrak{A}} = \llbracket t \rrbracket_{\zeta}^{\mathfrak{A}}; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \neg \varphi \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if } \llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = false; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \varphi \wedge \psi \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if } \llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = true \text{ and } \llbracket \psi \rrbracket_{\zeta}^{\mathfrak{A}} = true; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \varphi \vee \psi \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if } \llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = true \text{ or } \llbracket \psi \rrbracket_{\zeta}^{\mathfrak{A}} = true; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \varphi \rightarrow \psi \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if } \llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = false \text{ or } \llbracket \psi \rrbracket_{\zeta}^{\mathfrak{A}} = true; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \forall x \varphi \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if } \llbracket \varphi \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} = true \text{ for all } a \in \mathfrak{A} ; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \exists x \varphi \rrbracket_{\zeta}^{\mathfrak{A}}$	$:= \begin{cases} true, & \text{if there exists an } a \in \mathfrak{A} \text{ with } \llbracket \varphi \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} = true; \\ false, & \text{otherwise.} \end{cases}$

Table 1. The semantics of formulas.

Definition 4 (Firing of updates). The result of firing a consistent update set U in a state \mathfrak{A} is a new state $U(\mathfrak{A})$ with the same universe as \mathfrak{A} satisfying the following two conditions for the interpretations of function names f of Σ :

1. If $\langle f, a, b \rangle \in U$, then $f^{U(\mathfrak{A})}(a) = b$.
2. If there is no b with $\langle f, a, b \rangle \in U$ or if f is static, then $f^{U(\mathfrak{A})}(a) = f^{\mathfrak{A}}(a)$.

Since U is consistent, the state $U(\mathfrak{A})$ is determined in a unique way. Notice that only those locations can have a new value in state $U(\mathfrak{A})$ with respect to state \mathfrak{A} for which there is an update in U .

The composition ' $U ; V$ ' of two update sets U and V is defined such that the following equation is true for any state \mathfrak{A} :

$$(U ; V)(\mathfrak{A}) = V(U(\mathfrak{A}))$$

The equation says that applying the update set ' $U ; V$ ' to state \mathfrak{A} should be the same as first applying U and then V . Hence, ' $U ; V$ ' is the set of updates obtained from U by adding the updates of V and overwriting updates in U which are redefined in V . If U and V are consistent, then $U ; V$ is consistent, too.

Definition 5 (Composition of update sets). The composition of two update sets U and V is defined by $U ; V := \{ \langle f, a, b \rangle \in U \mid \neg \exists c \langle f, a, c \rangle \in V \} \cup V$.

In a given state, a transition rule of an ASM produces for each variable assignment an update set. Since the rule can contain recursive calls to other rules, it is also possible that the rule does not terminate and has no semantics at all.

$\overline{\llbracket \text{skip} \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright \emptyset}$		(skip)
$\overline{\llbracket f(s) := t \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright \{ \langle f, a, b \rangle \}}$	if $a = \llbracket s \rrbracket_{\zeta}^{\mathfrak{A}}$ and $b = \llbracket t \rrbracket_{\zeta}^{\mathfrak{A}}$	(upd)
$\frac{\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U \quad \llbracket S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright V}{\llbracket R S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U \cup V}$		(par)
$\frac{\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}{\llbracket \text{if } \varphi \text{ then } R \text{ else } S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}$	if $\llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = \text{true}$	(if ₁)
$\frac{\llbracket S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}{\llbracket \text{if } \varphi \text{ then } R \text{ else } S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}$	if $\llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = \text{false}$	(if ₂)
$\frac{\llbracket R \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} \triangleright U}{\llbracket \text{let } x = t \text{ in } R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}$	if $a = \llbracket t \rrbracket_{\zeta}^{\mathfrak{A}}$	(let)
$\frac{\llbracket R \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} \triangleright U_a \quad \text{for each } a \in I}{\llbracket \text{for all } x \text{ with } \varphi \text{ do } R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright \bigcup_{i \in I} U_i}$	if $I = \{a \in \mathfrak{A} : \llbracket \varphi \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} = \text{true}\}$	(forall)
$\frac{\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U \quad \llbracket S \rrbracket_{\zeta}^{U(\mathfrak{A})} \triangleright V}{\llbracket R ; S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U ; V}$	if U is consistent	(seq ₁)
$\frac{\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}{\llbracket R ; S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}$	if U is inconsistent	(seq ₂)
$\frac{\llbracket R \frac{t}{x} \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}{\llbracket \rho(t) \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U}$	if $\rho(x) = R$ is a rule definition	(def)

Table 2. Inductive definition of the semantics of ASM rules.

Therefore, the semantics of ASM rules is given by an inductive definition of a predicate $\mathcal{S}(R, \mathfrak{A}, \zeta, U)$ with the meaning ‘rule R yields in state \mathfrak{A} under the variable assignment ζ the update set U .’ Instead of $\mathcal{S}(R, \mathfrak{A}, \zeta, U)$ we write $\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U$ and present the inductive definition as a (possibly infinitary) calculus in Table 2. We say that a rule R is defined in state \mathfrak{A} under the variable assignment ζ , if there exists an update set U such that $\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U$ is derivable in the calculus in Table 2. Note that for each state \mathfrak{A} and variable assignment ζ there exists at most one update set U such that $\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U$ is derivable in the calculus. Hence transition rules are deterministic.

The notion of ASM run is the classical notion of computation of transition systems. A computation step in a given state consists in executing *simultaneously* all updates of the main transition rule of the ASM, if these updates are consistent. The run stops if the main transition rule is not defined or yields an inconsistent update set. If the update set is empty, then the ASM produces an infinite run (stuttering, never changing anymore the state). We do not allow that so-called *monitored* functions change during a computation.

Definition 6. (Run of an ASM) Let M be an ASM with vocabulary Σ , initial state \mathfrak{A} and main rule name ρ . Let ζ be a variable assignment. A *run* of M is a finite or infinite sequence $\mathfrak{B}_0, \mathfrak{B}_1, \dots$ of states for Σ such that the following conditions are satisfied:

1. $\mathfrak{B}_0 = \mathfrak{A}$.
2. If $\llbracket \rho \rrbracket_{\zeta}^{\mathfrak{B}_n}$ is not defined or inconsistent, then \mathfrak{B}_n is the last state.
3. Otherwise, $\mathfrak{B}_{n+1} = U(\mathfrak{B}_n)$, where $\llbracket \rho \rrbracket_{\zeta}^{\mathfrak{B}_n} \triangleright U$.

Runs are deterministic and independent of the variable assignment ζ , since we forbid global variables in rule definitions.

Remark 1. In the presence of sequential composition the following two rules are not equivalent:

$$\mathbf{let } x = t \mathbf{ in } R \quad \neq \quad R \frac{t}{x}.$$

For a counter example, consider the following transition rule:

$$\mathbf{let } x = f(0) \mathbf{ in } (f(0) := 1 ; f(1) := x)$$

If we substitute the term $f(0)$ for x , then we obtain:

$$f(0) := 1 ; f(1) := f(0)$$

In general, the two rules are not the same, because $f(0)$ is evaluated in different states. The following substitution property, however, is true for *static* terms t : If t is static and $\llbracket t \rrbracket_{\zeta}^{\mathfrak{A}} = a$, then

$$\llbracket R \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} \triangleright U \iff \llbracket R \frac{t}{x} \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U.$$

If the term t contains dynamic functions, then the equivalence is not necessarily true, because t could be evaluated in different states on the right-hand side (due to sequential compositions).

3 Formalizing the Consistency of ASMs

Following Groenboom and Renardel de Lavalette [4] we extend the language of first-order predicate logic by a modal operator $[R]$ for each rule R . The intended meaning of a formula $[R]\varphi$ is that the formula φ is true after firing R . More precisely, the formula $[R]\varphi$ is true iff one of the following conditions is satisfied:

1. R is not defined or the update set of R is inconsistent, or
2. R is defined, the update set of R is consistent and φ is true in the next state after firing the update set of R .

Equivalently we can say that the formula $[R]\varphi$ is true in state \mathfrak{A} under the variable assignment ζ iff for each set U such that $\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U$ is derivable and U is consistent, the formula φ is true in the state $U(\mathfrak{A})$ under ζ (see Table 3).

In order to express the definedness and the consistency of transition rules we extend the set of formulas by atomic formulas $\mathbf{def}(R)$ and $\mathbf{upd}(R, f, x, y)$. The

$\llbracket [R]\varphi \rrbracket_{\zeta}^{\mathfrak{A}} := \begin{cases} true, & \text{if } \llbracket \varphi \rrbracket_{\zeta}^{U(\mathfrak{A})} = true \text{ for each consistent } U \text{ with } \llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \text{def}(R) \rrbracket_{\zeta}^{\mathfrak{A}} := \begin{cases} true, & \text{if there exists an update set } U \text{ with } \llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U; \\ false, & \text{otherwise.} \end{cases}$
$\llbracket \text{upd}(R, f, s, t) \rrbracket_{\zeta}^{\mathfrak{A}} := \begin{cases} true, & \text{if ex. } U \text{ with } \llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U \text{ and } \langle f, \llbracket s \rrbracket_{\zeta}^{\mathfrak{A}}, \llbracket t \rrbracket_{\zeta}^{\mathfrak{A}} \rangle \in U; \\ false, & \text{otherwise.} \end{cases}$

Table 3. The semantics of modal formulas and basic predicates.

D1. $\text{def}(\text{skip})$	
D2. $\text{def}(f(s) := t)$	
D3. $\text{def}(R S) \leftrightarrow \text{def}(R) \wedge \text{def}(S)$	
D4. $\text{def}(\text{if } \varphi \text{ then } R \text{ else } S) \leftrightarrow (\varphi \wedge \text{def}(R)) \vee (\neg\varphi \wedge \text{def}(S))$	
D5. $\text{def}(\text{let } x = t \text{ in } R) \leftrightarrow \exists x (x = t \wedge \text{def}(R))$	if $x \notin \text{FV}(t)$
D6. $\text{def}(\text{forall } x \text{ with } \varphi \text{ do } R) \leftrightarrow \forall x (\varphi \rightarrow \text{def}(R))$	
D7. $\text{def}(R ; S) \leftrightarrow \text{def}(R) \wedge [R]\text{def}(S)$	
D8. $\text{def}(\rho(t)) \leftrightarrow \text{def}(R_x^t)$	if $\rho(x) = R$ is a rule definition of M

Table 4. Axioms for definedness.

U1. $\neg \text{upd}(\text{skip}, f, x, y)$	
U2. $\text{upd}(f(s) := t, f, x, y) \leftrightarrow s = x \wedge t = y, \quad \neg \text{upd}(f(s) := t, g, x, y) \quad \text{if } f \neq g$	
U3. $\text{upd}(R S, f, x, y) \leftrightarrow \text{def}(R S) \wedge (\text{upd}(R, f, x, y) \vee \text{upd}(S, f, x, y))$	
U4. $\text{upd}(\text{if } \varphi \text{ then } R \text{ else } S, f, x, y) \leftrightarrow (\varphi \wedge \text{upd}(R, f, x, y)) \vee (\neg\varphi \wedge \text{upd}(S, f, x, y))$	
U5. $\text{upd}(\text{let } z = t \text{ in } R, f, x, y) \leftrightarrow \exists z (z = t \wedge \text{upd}(R, f, x, y))$	if $z \notin \text{FV}(t)$
U6. $\text{upd}(\text{forall } z \text{ with } \varphi \text{ do } R, f, x, y) \leftrightarrow$ $\quad \text{def}(\text{forall } z \text{ with } \varphi \text{ do } R) \wedge \exists z (\varphi \wedge \text{upd}(R, f, x, y))$	
U7. $\text{upd}(R ; S, f, x, y) \leftrightarrow$ $\quad (\text{upd}(R, f, x, y) \wedge [R](\text{def}(S) \wedge \text{inv}(S, f, x))) \vee$ $\quad (\text{Con}(R) \wedge [R]\text{upd}(S, f, x, y))$	
U8. $\text{upd}(\rho(t), f, x, y) \leftrightarrow \text{upd}(R_z^t, f, x, y) \quad \text{if } \rho(z) = R \text{ is a rule definition of } M$	

Table 5. Axioms for updates.

semantics of these formulas is defined in Table 3 and the basic properties are listed in Tables 4 and 5. The formula $\text{def}(R)$ expresses that the rule R is defined. The formula $\text{upd}(R, f, x, y)$ expresses that rule R is defined and yields an update set which contains an update for f at x to y . The formula $\text{Con}(R)$ used in U7 to characterize an update of a sequential composition is defined as follows:

$$\text{Con}(R) := \text{def}(R) \wedge \bigwedge_{f \text{ dyn.}} \forall x, y, z (\text{upd}(R, f, x, y) \wedge \text{upd}(R, f, x, z) \rightarrow y = z)$$

It is true in a state iff the rule R is defined and yields a consistent update set:

$$\llbracket \text{Con}(R) \rrbracket_{\zeta}^{\mathfrak{A}} = \text{true} \iff \text{there exists a consistent } U \text{ with } \llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U.$$

The formula $\text{inv}(R, f, x)$ in U7 expresses that the rule R does not update the function f at the argument x . It is a simple abbreviation defined as follows:

$$\text{inv}(R, f, x) := \forall y \neg \text{upd}(R, f, x, y)$$

Note, that it would be wrong to define the predicate $\text{upd}(R, f, x, y)$ by saying that $f(x)$ is different from y in the present state but equal to y in the next state after firing rule R :

$$\text{upd}(R, f, x, y) := f(x) \neq y \wedge [R]f(x) = y \quad (\text{wrong definition})$$

Using this definition, the predicate $\text{upd}(f(0) := 1, f, 0, 1)$ would be false in a state where $f(0)$ is equal to 1, although the rule $f(0) := 1$ does update the function f at the argument 0 to 1.

4 Basic Axioms and Rules of the Logic

The formulas of the logic for abstract state machines are generated by the following grammar:

$$\begin{aligned} \varphi, \psi ::= & s = t \mid \neg \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \forall x \varphi \mid \exists x \varphi \mid \\ & \text{def}(R) \mid \text{upd}(R, f, s, t) \mid [R]\varphi \end{aligned}$$

A formula is called *pure* (or *first-order*), if it contains neither the predicate ‘def’ nor ‘upd’ nor the modal operator $[R]$. A formula is called *static*, if it does not contain dynamic function names. The formulas used in If-Then-Else and Forall rules must be pure formulas.

The semantics of formulas is given by the definitions in Table 1 and Table 3. The equivalence $\varphi \leftrightarrow \psi$ is defined by $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. A formula φ is called *valid*, if $\llbracket \varphi \rrbracket_{\zeta}^{\mathfrak{A}} = \text{true}$ for all states \mathfrak{A} and variable assignments ζ . The substitution of a term t for a variable x in a formula φ is denoted by $\varphi \frac{t}{x}$ and is defined as usual. Variables bound by a quantifier, a **let** or a **forall** have to be renamed when necessary. The substitution is also performed inside of transition rules that occur in formulas. The following substitution property holds.

Lemma 1 (Substitution). *If t is static and $a = \llbracket t \rrbracket_{\zeta}^{\mathfrak{A}}$, then $\llbracket \varphi \rrbracket_{\zeta \frac{a}{x}}^{\mathfrak{A}} = \llbracket \varphi \frac{t}{x} \rrbracket_{\zeta}^{\mathfrak{A}}$.*

We define two transition rules R and S to be *equivalent*, if they are equiconsistent and produce the same next state when they are fired.

Definition 7 (Equivalence). The formula $R \simeq S$ is defined as follows:

$$R \simeq S \text{ :} \iff (\text{Con}(R) \vee \text{Con}(S)) \rightarrow \text{Con}(R) \wedge \text{Con}(S) \wedge \\ \bigwedge_{f \text{ dyn.}} \forall x, y (\text{upd}(R, f, x, y) \rightarrow \text{upd}(S, f, x, y) \vee f(x) = y) \wedge \\ \bigwedge_{f \text{ dyn.}} \forall x, y (\text{upd}(S, f, x, y) \rightarrow \text{upd}(R, f, x, y) \vee f(x) = y)$$

The formula $R \simeq S$ has the intended meaning:

Lemma 2. *The formula $R \simeq S$ is true in \mathfrak{A} under ζ iff the following two conditions are true:*

1. $\llbracket \text{Con}(R) \rrbracket_{\zeta}^{\mathfrak{A}} = \text{true}$ iff $\llbracket \text{Con}(S) \rrbracket_{\zeta}^{\mathfrak{A}} = \text{true}$.
2. If $\llbracket R \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright U$, $\llbracket S \rrbracket_{\zeta}^{\mathfrak{A}} \triangleright V$ and U and V are consistent, then $U(\mathfrak{A}) = V(\mathfrak{A})$.

We already know that the axioms **D1–D8** and **U1–U8** are valid for a given abstract state machine M . Together with the following principles they will be the basic axioms and rules of our logic $\mathcal{L}(M)$.

I. Classical logic with equality: We use the axioms and rules of the classical predicate calculus with equality. The quantifier axioms, however, are restricted.

II. Restricted quantifier axioms:

1. $\forall x \varphi \rightarrow \varphi \frac{t}{x}$ if t is static or φ is pure
2. $\varphi \frac{t}{x} \rightarrow \exists x \varphi$ if t is static or φ is pure

III. Modal axioms and rules:

3. $[R](\varphi \rightarrow \psi) \wedge [R]\varphi \rightarrow [R]\psi$
4. $\frac{\varphi}{[R]\varphi}$
5. $\neg \text{Con}(R) \rightarrow [R]\varphi$
6. $\neg [R]\varphi \rightarrow [R]\neg \varphi$

IV. The Barcan axiom:

7. $\forall x [R]\varphi \rightarrow [R]\forall x \varphi$, if $x \notin \text{FV}(R)$.

V. Axioms for pure static formulas:

8. $\varphi \rightarrow [R]\varphi$ if φ is pure and static
9. $\text{Con}(R) \wedge [R]\varphi \rightarrow \varphi$ if φ is pure and static

VI. Axioms for def and upd:

10. **D1–D8** in Table 4
11. **U1–U8** in Table 5

VII. Update axioms for transition rules:

12. $\text{upd}(R, f, x, y) \rightarrow \text{def}(R)$
13. $\text{upd}(R, f, x, y) \rightarrow [R]f(x) = y$
14. $\text{inv}(R, f, x) \wedge f(x) = y \rightarrow [R]f(x) = y$

VIII. Extensionality axiom for transition rules:

15. $R \simeq S \rightarrow ([R]\varphi \leftrightarrow [S]\varphi)$

IX. Axioms from dynamic logic:

16. $[\text{skip}]\varphi \leftrightarrow \varphi$
17. $[R; S]\varphi \leftrightarrow [R][S]\varphi$

The principles I–IX are valid, therefore the logic is sound.

Theorem 1 (Soundness). *If a formula is derivable with the axioms and rules I–IX, then it is valid.*

The formula $\forall x \varphi \rightarrow \varphi_x^t$ is not valid for non-static terms t . Consider the following tautology:

$$\forall x (x = 0 \rightarrow [f(0) := 1]x = 0).$$

If we substitute the term $f(0)$ for x , then we obtain the formula

$$f(0) = 0 \rightarrow [f(0) := 1]f(0) = 0.$$

This formula is not valid. Hence, the quantifier axioms must be restricted.

Lemma 3. *The following consistency properties are derivable:*

18. $\text{Con}(\text{skip})$
19. $\text{Con}(f(s) := t)$
20. $\text{Con}(R S) \leftrightarrow \text{Con}(R) \wedge \text{Con}(S) \wedge \text{joinable}(R, S)$
21. $\text{Con}(\text{if } \varphi \text{ then } R \text{ else } S) \leftrightarrow (\varphi \wedge \text{Con}(R)) \vee (\neg\varphi \wedge \text{Con}(S))$
22. $\text{Con}(\text{let } x = t \text{ in } R) \leftrightarrow \exists x (x = t \wedge \text{Con}(R))$ *if* $x \notin \text{FV}(t)$
23. $\text{Con}(\text{forall } x \text{ with } \varphi \text{ do } R) \leftrightarrow \forall x (\varphi \rightarrow \text{Con}(R)) \wedge \forall y (\varphi_x^y \rightarrow \text{joinable}(R, R_x^y))$
24. $\text{Con}(R; S) \leftrightarrow \text{Con}(R) \wedge [R]\text{Con}(S)$
25. $\text{Con}(\rho(t)) \leftrightarrow \text{Con}(R_x^t)$ *if* $\rho(x) = R$ is a rule definition of M

The predicate $\text{joinable}(R, S)$ which is used in 20 to reduce the consistency of a parallel composition $R S$ into consistency properties of R and S is defined as follows (where x, y, z are not free in R):

$$\text{joinable}(R, S) := \bigwedge_{f \text{ dyn.}} \forall x, y, z (\text{upd}(R, f, x, y) \wedge \text{upd}(S, f, x, z) \rightarrow y = z),$$

It expresses that the update sets of R and S do not conflict. This means, whenever R and S both update a function f at the same argument x , then the new values of f at x are the same.

Lemma 4. *The following principles are derivable:*

26. $\text{Con}(R) \wedge [R]f(x) = y \rightarrow \text{upd}(R, f, x, y) \vee (\text{inv}(R, f, x) \wedge f(x) = y)$

27. $\text{Con}(R) \wedge [R]\varphi \rightarrow \neg[R]\neg\varphi$
 28. $[R]\exists x \varphi \leftrightarrow \exists x [R]\varphi,$ *if $x \notin \text{FV}(R)$.*

Groenboom and Renardel de Lavalette introduce in [4] different axioms for transition rules. Their axioms FM1, FM2, AX1, AX2 are derivable in our system using the update axioms 13 and 14.

Lemma 5. *The following principles of [4] are derivable:*

29. $s = x \rightarrow (y = t \leftrightarrow [f(s) := t]f(x) = y)$
 30. $s \neq x \rightarrow (y = f(x) \leftrightarrow [f(s) := t]f(x) = y)$
 31. $[R]f(x) = y \wedge [S]f(x) = y \rightarrow [R S]f(x) = y$
 32. $f(x) \neq y \wedge ([R]f(x) = y \vee [S]f(x) = y) \rightarrow [R S]f(x) = y.$

The following inverse implication of 31 and 32 is not mentioned in [4] (maybe because of the lack of a consistency notion), but is derivable in our system:

$$\text{Con}(R S) \wedge [R S]f(x) = y \rightarrow \\ ([R]f(x) = y \wedge [S]f(x) = y) \vee (f(x) \neq y \wedge ([R]f(x) = y \vee [S]f(x) = y))$$

Several principles known from dynamic logic are derivable using the extensionality axiom 15.

Lemma 6. *The following principles are derivable:*

33. **if φ then R else S** $\psi \leftrightarrow (\varphi \wedge [R]\psi) \vee (\neg\varphi \wedge [S]\psi)$
 34. **let $x = t$ in R** $\varphi \leftrightarrow \exists x (x = t \wedge [R]\varphi),$ *if $x \notin \text{FV}(t) \cup \text{FV}(\varphi)$.*
 35. $[\rho(t)]\varphi \leftrightarrow [R_{\frac{t}{x}}]\varphi,$ *if $\rho(x) = R$ is a rule definition of M .*

Schönege uses in his sequent calculus for the extended dynamic logic in [11] new rules that express the commutativity, the associativity and similar properties of the parallel combination of transition rules. In our system, these properties are derivable.

Lemma 7. *The following principles of [11] are derivable:*

36. $(R \text{ skip}) \simeq R$
 37. $(R S) \simeq (S R)$
 38. $((R S) T) \simeq (R (S T))$
 39. $(R R) \simeq R$
 40. **(if φ then R else S) T** \simeq **if φ then $(R T)$ else $(S T)$**
 41. **T (if φ then R else S)** \simeq **if φ then $(T R)$ else $(T S)$**

If we can derive $R \simeq S$, then we immediately obtain the principle $[R]\varphi \leftrightarrow [S]\varphi$ using the extensionality axiom 15. It is not clear to us, whether for example the commutativity of the parallel composition, $[R S]\varphi \leftrightarrow [S R]\varphi$, could be derived in the formal system of [4].

Lemma 8. *The following properties of the sequential composition are derivable:*

42. $(R ; \text{skip}) \simeq R$
 43. $(\text{skip} ; R) \simeq R$

44. $((R; S); T) \simeq (R; (S; T))$
 45. $(\text{if } \varphi \text{ then } R \text{ else } S); T \simeq \text{if } \varphi \text{ then } (R; T) \text{ else } (S; T)$

The dynamic logic with array assignments (see [6]) uses a substitution principle which is derivable in our system. Let φ be a pure (first-order) formula. Then by $\varphi_{\frac{t}{f(s)}}$ we denote the formula which is obtained in the following way. First, φ is transformed into an equivalent formula

$$\varphi \leftrightarrow \exists \mathbf{x} \exists \mathbf{y} \left(\bigwedge_{i=1}^n f(x_i) = y_i \wedge \psi \right),$$

where $\mathbf{x} = x_1, \dots, x_n$, $\mathbf{y} = y_1, \dots, y_n$ and ψ does not contain f . Then we define:

$$\varphi_{\frac{t}{f(s)}} := \exists \mathbf{x} \exists \mathbf{y} \left(\bigwedge_{i=1}^n ((x_i = s \wedge y_i = t) \vee (x_i \neq s \wedge f(x_i) = y_i)) \wedge \psi \right)$$

Lemma 9. *For any first-order formula φ , the following substitution principle is derivable: $\varphi_{\frac{t}{f(s)}} \leftrightarrow [f(s) := t]\varphi$.*

The If-Then rule can be defined in terms of If-Then-Else in the standard way:

$$\text{if } \varphi \text{ then } R := \text{if } \varphi \text{ then } R \text{ else skip}$$

An ASM is called *simple*, if it is defined by a single rule R , which has the following form:

$$\begin{aligned} &\text{if } \varphi_1 \text{ then } f(s_1) := t_1 \\ &\text{if } \varphi_2 \text{ then } f(s_2) := t_2 \\ &\quad \vdots \\ &\text{if } \varphi_n \text{ then } f(s_n) := t_n \end{aligned}$$

Simple ASMs have the obvious properties formulated in the following lemma. Property 49 is a variant of the basic axiom of Poetzsch-Heffter [8]. It can easily be extended to disjoint If-Then rules with simultaneous function updates.

Lemma 10. *Let R be the rule of a simple ASM. Then,*

46. $\text{Con}(R) \leftrightarrow \bigwedge_{i < j} (\varphi_i \wedge \varphi_j \wedge s_i = s_j \rightarrow t_i = t_j)$
 47. $\text{upd}(R, f, x, y) \leftrightarrow \bigvee_{i=1}^n (\varphi_i \wedge x = s_i \wedge y = t_i)$
 48. $\text{inv}(R, f, x) \leftrightarrow \bigwedge_{i=1}^n (\varphi_i \rightarrow x \neq s_i)$
 49. $\bigvee_{i=1}^n \varphi_i \wedge \bigwedge_{i < j} \neg(\varphi_i \wedge \varphi_j) \wedge \bigwedge_{i=1}^n (\varphi_i \rightarrow \psi_{\frac{t_i}{f(s_i)}}) \rightarrow [R]\psi, \quad \text{if } \psi \text{ is first-order.}$

Iteration can be reduced to recursion. We can define the While rule recursively, as follows:

$$\text{while } \varphi \text{ do } R = \text{if } \varphi \text{ then } (R; \text{while } \varphi \text{ do } R)$$

The expression **while** φ **do** R has to be read as a rule call $\rho(\mathbf{x})$, where \mathbf{x} are the free variables of φ and R . So the above equation stands for the following rule definition:

$$\rho(\mathbf{x}) = \mathbf{if} \varphi \mathbf{then} (R; \rho(\mathbf{x}))$$

Lemma 11. *The following properties of the While rule are derivable:*

$$50. \text{Con}(\mathbf{while} \varphi \mathbf{do} R) \leftrightarrow (\varphi \rightarrow \text{Con}(R) \wedge [R]\text{Con}(\mathbf{while} \varphi \mathbf{do} R))$$

$$51. [\mathbf{while} \varphi \mathbf{do} R]\psi \leftrightarrow (\varphi \wedge [R][\mathbf{while} \varphi \mathbf{do} R]\psi) \vee (\neg\varphi \wedge \psi)$$

Several properties of ASMs can be expressed using the basic logic (where M is the distinguished rule name of the ASM and φ_{init} is a formula characterizing initial states):

- The formula ψ ensures consistency: $(\varphi_{\text{init}} \rightarrow \psi) \wedge (\psi \rightarrow \text{Con}(M) \wedge [M]\psi)$.
- The formula ψ is an invariant: $(\varphi_{\text{init}} \rightarrow \psi) \wedge (\psi \rightarrow [M]\psi)$.

The statement in [13] for the correctness of the compiler from Java to the JVM can be formulated as follows:

$$(\varphi_{\text{init}} \rightarrow \varphi_{\text{eqv}}) \wedge (\varphi_{\text{eqv}} \rightarrow [J](\varphi_{\text{eqv}} \vee [V]\varphi_{\text{eqv}} \vee [V][V]\varphi_{\text{eqv}} \vee [V][V][V]\varphi_{\text{eqv}}))$$

Here, J is an ASM that specifies the semantics of a Java source level program according to the *Java Language Specification*; V is an ASM that specifies the *Java Virtual Machine*. The two ASMs have disjoint dynamic function names and use the same static functions. The formula φ_{eqv} expresses that two dynamic states of the two ASMs are equivalent for a given Java program and its compiled bytecode program. The above formula says, that if two states are equivalent, then for each step of J the ASM V has to make zero, one, two, or three steps to reach an equivalent state again. The proof in [13] which comprises 83 cases could be carried out in the basic system with appropriate structural induction principles for lists and abstract syntax trees (which are encoded using static functions).

5 Completeness for hierarchical ASMs

An ASM is called *hierarchical*, if the call graph of the rule definitions does not contain cycles, in other words, if the ASM does not contain recursive rule definitions. An ASM is hierarchical iff it is possible to assign levels to the rule names such that in a rule definition $\rho(x) = R$ the levels of rule names in R are less than the level of ρ . Transition rules of hierarchical ASMs are always defined. If R is a transition rule which uses rules from a hierarchical machine M , then $\text{def}(R)$ is derivable in $\mathcal{L}(M)$.

Theorem 2 (Completeness). *If M is a hierarchical ASM and φ is valid, then φ is derivable in $\mathcal{L}(M)$.*

The proof of the completeness theorem follows the traditional Henkin-style completeness proof and uses the fact that for a maximal consistent set Φ of formulas, if $\text{Con}(R) \in \Phi$, then the set $\{\psi \mid [R]\psi \in \Phi\}$ is also maximal consistent. The extensionality axiom 15, Axiom 16 for **skip** and Axiom 17 for the sequential compositions are not used in the completeness proof. Since the axioms are valid, by the completeness theorem they must be derivable for hierarchical ASMs.

Acknowledgment We are grateful to Egon Börger for helpful comments on an earlier version of the article. Because of his remarks we decided not to use $\text{Con}(R)$, $\text{upd}(R, f, x, y)$ and $\text{inv}(R, f, x)$ as basic notions of the logic, but $\text{def}(R)$ and $\text{upd}(R, f, x, y)$ instead.

References

1. E. Börger and J. Huggins. Abstract State Machines 1988–1998: Commented ASM Bibliography. *Bulletin Europ. Assoc. Theoret. Comp. Science*, 64:105–127, 1998. Updated bibliography available at <http://www.eecs.umich.edu/gasm>.
2. E. Börger and J. Schmid. Composition and submachine concepts. In P. Clote and H. Schwichtenberg, editors, *Computer Science Logic (CSL 2000)*, pages 41–60. Springer-Verlag, Lecture Notes in Computer Science 1862, 2000.
3. A. Gargantini and E. Riccobene. Encoding abstract state machines in PVS. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, pages 303–322. Springer-Verlag, Lecture Notes in Computer Science 1912, 2000.
4. R. Groenboom and G. R. Renardel de Lavalette. A formalization of evolving algebras. In *Proceedings of Accolade 95*. Dutch Research School in Logic, 1995.
5. Y. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1993.
6. D. Harel. Dynamic logic. In D. M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, pages 497–604. Reidel, Dordrecht, 1983.
7. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press, 2000.
8. A. Poetzsch-Heffter. Deriving partial correctness logics from evolving algebras. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 434–439. Elsevier, Amsterdam, 1994.
9. G. Schellhorn. *Verification of Abstract State Machines*. PhD thesis, Universität Ulm, 1999.
10. G. Schellhorn and W. Ahrendt. Reasoning about abstract state machines: the WAM case study. *J. of Universal Computer Science*, 3(4):377–413, 1997.
11. A. Schönegege. Extending dynamic logic for reasoning about evolving algebras. Technical Report IRATR–1995–49, Universität Karlsruhe, Institut für Logik, Komplexität und Deduktionssysteme, 1995.
12. N. Shankar. Symbolic analysis of transition systems. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, pages 287–302. Springer-Verlag, Lecture Notes in Computer Science 1912, 2000.
13. R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine—Definition, Verification, Validation*. Springer-Verlag, 2001.
14. J. van Benthem and J. A. Bergstra. Logic of transition systems. *J. of Logic, Language, and Information*, 3(4):247–283, 1995.