

A Semantic Characterization of Unbounded-Nondeterministic Abstract State Machines

Andreas Glausch and Wolfgang Reisig

Humboldt-Universität zu Berlin
Institut für Informatik
{glausch,reisig}@informatik.hu-berlin.de

Abstract. Universal algebra usually considers and examines algebras as static entities. In the mid 80ies Gurevich proposed *Abstract State Machines* (ASMs) as a computation model that regards algebras as dynamic: a state of an ASM is represented by a freely chosen algebra which may change during a computation. In [8] Gurevich characterizes the class of *sequential ASMs* in a purely semantic way by five amazingly general and elegant axioms. In [9] this result is extended to *bounded-nondeterministic ASMs*.

This paper considers the general case of *unbounded-nondeterministic ASMs*: in each step, an unbounded-nondeterministic ASM may choose among unboundedly many (sometimes infinitely many) alternatives. We characterize the class of unbounded-nondeterministic ASMs by an extension of Gurevich's original axioms for sequential ASMs. We apply this result to prove the reversibility of unbounded-nondeterministic ASMs.

1 Introduction

Abstract State Machines (ASMs) have been introduced as a “computational model that is more powerful and more universal than the standard computational models” by Yuri Gurevich in 1985 [6]. This is achieved by a combination of two classic notions of computer science and mathematics: *transition systems* and *algebras*.

Transition systems play a fundamental role in theoretical computer science. Usually, the operational semantics of a discrete computational model is specified in terms of a transition system, consisting of a set of *states* and a *next-state relation*. Examples are the transition systems as generated by Turing machines, λ -expressions, Petri nets, etc. The computational model of ASMs also fits into this setting, but differs from classical computational models in its general notion of states: each state of an ASM is an algebra.

As usual, an algebra A comprises a nonempty set U_A (its *universe*) together with finitely many functions defined over U_A , each with a fixed arity. No additional properties are required. In fact, *every* algebra may serve as a state of an ASM. As a consequence, a state of an ASMs may naturally include any

mathematical data structure that can be described in terms of logic, e.g. sets, real numbers, abstract geometrical objects, vector spaces, or even uncomputable functions. In classical computational models, such concepts usually require a particular encoding, if possible at all.

The seamless integration of arbitrary data structures make ASMs particularly well suited for abstract and natural modeling of algorithms and systems in general. Accordingly, ASMs have been extended to a full-fledged design and analysis methodology [5, 4]. By stepwise refinement and composition of ASMs, large-scale real-world systems have been formally modeled and analyzed [10, 12].

2 Scope and Contribution of this Paper

Classically, ASMs employ a pseudo-code like program syntax to describe the updates to be applied to a state A . This syntax is based on Σ -terms defined over the signature Σ of A and some additional elementary control structures. During the last decade, special attention towards a *syntax-independent* characterization of ASMs has been paid. Such characterizations are valuable in order to compare the expressive power of different variants of ASMs with classical computation models, and help to identify subtle properties of ASMs.

As stated above, ASMs fit into the general setting of transition systems. Correspondingly, a class \mathcal{C} of ASMs may be characterized by answering the following question:

Which transition systems can be represented by the ASMs in \mathcal{C} ?

In [8] Gurevich answers this question for the class of *sequential small-step ASMs* in a surprisingly elegant way: he defines a class of transition systems which he calls *sequential algorithms* by three general and semantic axioms, and proves this class to be equivalent to sequential ASMs (c.f. also [11]). Later, Blass and Gurevich identified similar characterizations for other variants of ASMs, including bounded-nondeterministic, parallel, and interactive versions [9, 1–3].

In this paper we contribute to this work by a corresponding result for the general case of *unbounded-nondeterministic small-step ASMs* (nondeterministic ASMs for short). Nondeterministic ASMs have been introduced in [7] as “ASMs with qualified choose”. We define the class of *nondeterministic algorithms* by purely semantic axioms, and prove that this class is equivalent to nondeterministic ASMs. This result shows that nondeterministic ASMs capture a surprisingly large class of transition systems.

We exemplify the profit of this result by proving the reversibility of nondeterministic ASMs: for each nondeterministic ASM M there exists a nondeterministic ASM M^{-1} executing M in reverse order.

The rest of this paper is organized as follows. In the next section we axiomatize the class of nondeterministic algorithms. In Sect. 4 we exemplify and define nondeterministic ASMs. Finally, we prove a theorem stating the equivalence of both notions, and apply this theorem in order to show the reversibility of nondeterministic ASMs.

3 Nondeterministic Algorithms

In [8] Gurevich defines the class of *sequential algorithms* by the following (slightly rearranged) five axioms which are very general, nevertheless simple and intuitive:

1. A sequential algorithm consists of a set of states \mathcal{S} , a set of initial states $\mathcal{J} \subseteq \mathcal{S}$, and a next-state function $\tau : \mathcal{S} \rightarrow \mathcal{S}$.
2. Each state $A \in \mathcal{S}$ is an algebra.
3. τ preserves the universe of states.
4. \mathcal{S} and \mathcal{J} are closed under isomorphism, and τ preserves isomorphism.

The decisive fifth axiom of sequential algorithms is *bounded exploration*. Intuitively, the bounded exploration axiom reads:

5. A finite set of ground terms is sufficient to characterize τ .

Gurevich shows in [8] that each sequential algorithm can be represented syntactically by a corresponding sequential ASM.

In this section we define the class of *nondeterministic algorithms* by five likewise simple axioms. The first four axioms are a canonical nondeterministic extension of Gurevich's original axioms. This extension has been presented in [9] already. The fifth axiom is entirely new: intuitively, it only requires a nondeterministic algorithm to perform a bounded amount of work in each step. In the following subsections, we present the axioms and justify their reasonability.

3.1 A Nondeterministic Algorithm Constitutes a Transition System

As indicated in the introduction already, the operational behaviour of a discrete algorithm specified in a particular computation model is usually given by a transition system. Therefore, we assume that the behaviour of every nondeterministic algorithm can naturally be represented by a nondeterministic transition system.

Axiom N1 (states and transitions) *A nondeterministic algorithm \mathcal{N} consists of*

- a set of states $S_{\mathcal{N}}$,
- a set of initial states $I_{\mathcal{N}} \subseteq S_{\mathcal{N}}$
- a next-state relation $\rightarrow_{\mathcal{N}} \subseteq S_{\mathcal{N}} \times S_{\mathcal{N}}$.

Each pair $(A, A') \in \rightarrow_{\mathcal{N}}$ is a *step* of \mathcal{N} . As usual, a *run* of \mathcal{N} is a sequence $A_0 A_1 A_2 \dots$ of states with $A_0 \in I_{\mathcal{N}}$ and $A_i \rightarrow_{\mathcal{N}} A_{i+1}$ for all indices i .

3.2 A State of a Nondeterministic Algorithm is an Algebra

The huge experience on algebraic specification confirms that algebras are general enough to faithfully describe any static mathematical entity on any level of abstraction. Consequently, it is legitimate to assume that every state of every

conceivable algorithm can be naturally described by an algebra. The second axiom formalizes this idea.

As usual, a *signature* Σ is used to address the functions of an algebra: Σ consists of finitely many function symbols f_1, \dots, f_k , each f_i with its arity n_i . An algebra A is a Σ -*algebra* if A determines for each n -ary function symbol f a unique n -ary function f_A .

As an algorithm always has a finite syntactical representation, an algorithm addresses only a finite set of functions. Hence, a single signature (with a finite set of symbols) suffices for all states. This leads to the second axiom:

Axiom N2 (states are algebras) *For a nondeterministic algorithm N , all states in S_N are algebras over the same signature Σ_N .*

Due to this axiom, we use the notions *state* and *algebra* interchangeably in this paper. As a running example we consider the following algebra Q , with universe $U_Q = \{1, 2, 3\}$, consisting of two nullary functions a_Q and b_Q , and two unary functions v_Q and next_Q :

$$\begin{array}{llll} a_Q = 1 & b_Q = 2 & v_Q(1) = 1 & \text{next}_Q(1) = 2 \\ & & v_Q(2) = 2 & \text{next}_Q(2) = 3 \\ & & v_Q(3) = 3 & \text{next}_Q(3) = 1 \end{array}$$

Hence, the signature Σ_Q of Q consists of the nullary function symbols a and b , and the unary function symbols v and next .

3.3 Steps of a Nondeterministic Algorithm Preserve the Universe

The universe U_A of an algebra A comprises all elementary semantic objects of A . In addition, A defines relationships between these objects in terms of functions over U_A . In this sense, the elements of U_A are atomic and foundational objects that cannot be decomposed, destroyed, or created. Only the functions of A are modified. As an example consider the Euclidian algorithm which computes the greatest common divisor of two given integers. The states of the Euclidian algorithm are built over the universe of all integers. A computation of the Euclidian algorithm does neither add nor remove integers from this universe. It merely computes new relationships such as “3 is the greatest common divisor of 12 and 27”. Consequently, the third axiom reads:

Axiom N3 (universe preservation) *For a nondeterministic algorithm N the following holds: for each step (A, A') of N , A and A' have the same universe.*

3.4 Steps of a Nondeterministic Algorithm Preserve Isomorphisms

As usual, an isomorphism i between two Σ -algebras A and B is a bijective mapping $i : U_A \rightarrow U_B$ that preserves the functions of A . Isomorphic algebras only differ in their concrete representation of the universe, whereas the functions of both algebras are essentially the same.

For an algorithm the concrete representation of the universe is inessential. For example, the Euclidean algorithm computes the greatest common divisor regardless whether the integers are represented by some transistors on a chip or by ink on paper. In general, an algorithm does not distinguish isomorphic states, but computes isomorphic next-states at isomorphic states. This insight is formalized by the fourth axiom:

Axiom N4 (isomorphism preservation) *For a nondeterministic algorithm \mathcal{N} the following holds:*

- (i) $\mathcal{S}_{\mathcal{N}}$ and $\mathcal{J}_{\mathcal{N}}$ are closed under isomorphism.
- (ii) Let (A, A') be a step of \mathcal{N} and let $B \in \mathcal{S}_{\mathcal{N}}$ with an isomorphism $i : A \rightarrow B$. Then there is a step (B, B') of \mathcal{N} such that $i : A' \rightarrow B'$ is an isomorphism.

Alternatively, (ii) may be formulated based on the well-known notion of *simulation relation*: each bijective mapping i induces a simulation relation \simeq on the states of \mathcal{N} that is defined as $A \simeq B$ iff i is an isomorphism from A to B . Hence, isomorphic states simulate each other.

3.5 Steps of a Nondeterministic Algorithm Perform Bounded Work

The axioms N1–N4 are merely Gurevich’s classical first four axioms to sequential algorithms, adjusted to the nondeterministic case. The fifth axiom presented next is new and requires some additional notions.

A real-world processor (e.g. a computer, an organization, or a human being) executing an algorithm performs only a *bounded* amount of work in each step (algorithms that allow for *unbounded* amount of work in each step are considered in [1]). Therefore, it is quite natural to require an algorithm to limit the amount of work to be done in each step. In the following we formalize this vague idea.

According to the previous axioms, a step of a nondeterministic algorithm preserves the signature and the universe of a state. That is, for a step (A, A') , A and A' share the same function symbols and the same function arguments, and differ only in their function values. In order to represent such differences formally, it is useful not to consider A as a collection of functions, but as a set of location-value-triples. A *location* of A consists of a n -ary function symbol \mathbf{f} and a n -ary argument tuple \bar{a} . For example $(\mathbf{v}, [1])$ is a location of the state Q (we enclose the argument tuple in square brackets for the sake of readability). Each location (\mathbf{f}, \bar{a}) of A defines a unique value $v = \mathbf{f}_A(\bar{a})$. The triple (\mathbf{f}, \bar{a}, v) represents a small component of A which we call a *molecule* of A . For example, $(\mathbf{v}, [1], 1)$ is a molecule of the state Q . Intuitively, the molecule $(\mathbf{v}, [1], 1)$ states that “the function denoted by \mathbf{v} maps the argument tuple $[1]$ to the value 1”.

A state A is completely described by its set of molecules. For example, the above state Q is represented by the following set of molecules:

$$Q = \{ (\mathbf{a}, [], 1), (\mathbf{b}, [], 2), \\ (\mathbf{v}, [1], 1), (\mathbf{v}, [2], 2), (\mathbf{v}, [3], 3), \\ (\mathbf{next}, [1], 2), (\mathbf{next}, [2], 3), (\mathbf{next}, [3], 1) \}.$$

Calling them “updates”, Gurevich employed molecules already in [7] to describe differences between algebras.

As announced above we intend to formalize the idea of “performing bounded work in each step”: only a bounded part of a state A should contribute to a step, whereas the rest of A remains unaffected. The representation of A by a set of molecules permits a simple formalization of “a part of A ”: each subset $M \subseteq A$ is a *substate* of A . As an example, the set

$$M_Q =_{\text{def}} \{ (\mathbf{a}, [], 1), (\mathbf{v}, [2], 2) \}.$$

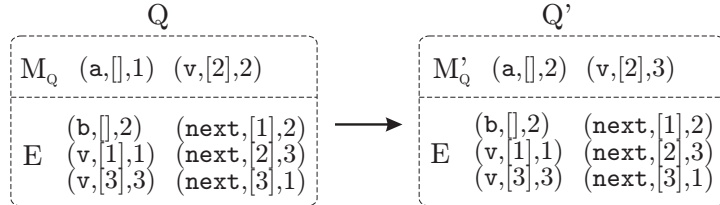
is a substate of Q .

Substates are used to describe steps that involve only a bounded part of a state: a *substep* changes a substate M by updating the values of the molecules in M . For instance, with

$$M'_Q =_{\text{def}} \{ (\mathbf{a}, [], 2), (\mathbf{v}, [2], 3) \}$$

the pair (M_Q, M'_Q) is a substep which changes the substate M_Q to the substate M'_Q . In general, a substep is a pair of substates (M, M') such that the locations of the molecules of M and M' coincide (i.e. M and M' differ only in the values of their molecules).

We employ substeps to capture the “amount of work” performed by a step of \mathcal{N} : each step (A, A') is decomposed into a substep (M, M') and a substate E disjoint from M and M' such that $(A, A') = (M \cup E, M' \cup E)$. Intuitively, the substep (M, M') describes the actual state change performed by the step whereas E describes the part of the state that is unaffected by the step (we use the letter “ E ” for “external”). In this case, we call the step (A, A') a *completion* of (M, M') . The following figure shows a step (Q, Q') which is a completion of the substep (M_Q, M'_Q) :



A “bounded amount of work” then is captured by a substep bounded in size: For a natural number k , a substep (M, M') is *k-bounded* iff $|M| \leq k$ (which is equivalent to $|M'| \leq k$). We are now able to formulate the final axiom stating that the steps of a nondeterministic algorithm perform only bounded substeps:

Axiom N5 (bounded work) *For a nondeterministic algorithm \mathcal{N} there exists a constant $k \in \mathbb{N}$ and a set \mathcal{W} of k -bounded substeps such that for all states A, A' of \mathcal{N} the following holds: (A, A') is a step of \mathcal{N} iff (A, A') is an completion of a substep in \mathcal{W} .*

As \mathcal{W} witnesses the fact that \mathcal{N} performs only a bounded amount of work in each step, we call \mathcal{W} a *bounded-work witness* of \mathcal{N} .

Though every substep in \mathcal{W} is bounded, the set \mathcal{W} itself may be infinite, even uncountably infinite. For example, consider a function symbol \mathbf{r} holding a real number: \mathcal{W} may contain uncountably many substeps that change the value of \mathbf{r} to a different real number. This points out a decisive difference to Gurevich's original bounded-exploration axiom: a finite set of ground terms is not sufficient to characterize such rich behaviour.

We believe that the Axioms N1–N5 are intuitively convincing requirements to discrete nondeterministic algorithms. However, we do not demand any further requirements: we call *any* entity satisfying the Axioms N1–N5 a nondeterministic algorithm.

4 Nondeterministic Abstract State Machines

In the previous section we introduced the class of nondeterministic algorithms in a purely semantic and declarative way. This may appear strange, as algorithms usually are represented in an explicit and syntactical form, e.g. by program code. This raises the question whether a given nondeterministic algorithm can be represented in a syntactical way at all: is there a language expressive enough to describe *any* nondeterministic algorithm? In the following we answer this questions positively by presenting the operational computation model of *nondeterministic ASMs*, which has been introduced in [7] already.

We start by introducing the syntax and semantics of *nondeterministic ASM rules* which divide into four rule types: *assignment rules*, *conditional rules*, *parallel rules*, and *choice rules*. The syntax of these rules is simple and intuitive, and reminds of pseudo-code. However, in contrast to pseudo-code, nondeterministic ASM rules have a formal semantics. Nondeterministic ASM rules form the syntactical basis of nondeterministic ASMs introduced afterwards.

4.1 Assignment Rules

As usual, Σ -terms are constructed inductively from a signature Σ and a set of variables V . Given a Σ -algebra A and a variable assignment $\alpha : V \rightarrow U_A$, each Σ -term t is evaluated to a unique value $t_{A,\alpha} \in U_A$. We may skip the index α in case t is ground (i.e. t contains no variables).

Terms are used to form *assignment rules* which update a single function value of an algebra. An example built from signature Σ_Q is the assignment rule EXASSIGN:

$$\mathbf{v}(\mathbf{a}) := \mathbf{next}(\mathbf{b}).$$

Executing rule EXASSIGN at state Q assigns to the function symbol \mathbf{v} at the argument $a_Q = 1$ the value $\mathbf{next}(\mathbf{b})_Q = 3$ (as all terms in EXASSIGN are ground, a variable assignment α is not required). The result is a new state Q' equal to Q except for the value at location $(\mathbf{v}, [1])$.

The general form of an assignment rule ASSIGN is

$$\mathbf{f}(t_1, \dots, t_n) := t'$$

where t_1, \dots, t_n and t' are Σ -terms, and \mathbf{f} is a n -ary function symbol of Σ . Applied at a state A and a variable assignment α , ASSIGN updates the value of the function symbol \mathbf{f} at the argument $\bar{a} =_{\text{def}} (t_{1A,\alpha}, \dots, t_{nA,\alpha})$ by the value $v =_{\text{def}} t'_{A,\alpha}$. This update is represented by an *update molecule* (\mathbf{f}, \bar{a}, v) .

In general, ASM rules may update more than a single location of A . Multiple updates are represented by a *set* of update molecules, which is called an *update set*. The update set of the assignment rule ASSIGN at state A and variable assignment α is defined as the singleton set

$$\text{ASSIGN}_{A,\alpha} =_{\text{def}} \{ (\mathbf{f}, \bar{a}, v) \}$$

with \bar{a} and v as defined above. Such an update set is applied to A by changing the function values of A according to the update molecules in the update set. The formal definition is straightforward and will be given in Sect. 4.3.

4.2 Conditional Rules

An assignment rule may be guarded by a condition, which is represented by a *conditional rule*. For example, the conditional rule EXCOND

$$\text{if } (\mathbf{a}=\mathbf{b}) \text{ then } \mathbf{a}:=\text{next}(\mathbf{a})$$

executes the assignment rule at a state A only if the condition $\mathbf{a}=\mathbf{b}$ holds in A .

The condition may be an arbitrary Boolean formula. In technical terms, a Boolean formula ϕ consists of several *term equations* of the form $t_1=t_2$ connected by the usual Boolean operations \neg , \wedge , and \vee . For a given state A and a variable assignment α , the truth value of ϕ is computed in the obvious way.

The general form of a conditional rule COND is

$$\text{if } \phi \text{ then ASSIGN}$$

where ϕ is a Boolean formula and ASSIGN is an assignment rule. Its semantics is obvious: ASSIGN is executed if the condition ϕ is satisfied by A and α . Otherwise the state A is left unchanged. Hence, the update set of COND at A and α is defined as

$$\text{COND}_{A,\alpha} =_{\text{def}} \begin{cases} \text{ASSIGN}_{A,\alpha} & , \text{if } \phi \text{ is satisfied by } A \text{ and } \alpha \\ \emptyset & , \text{otherwise.} \end{cases}$$

For technical convenience, we assume every assignment rule as a special conditional rule whose condition holds in every state.

4.3 Parallel Rules

Several conditional rules may be executed simultaneously, which is represented by a *parallel rule*. A simple example is the parallel rule EXPAR

par a:=b b:=a endpar.

EXPAR simultaneously executes both assignment rules (which are special conditional rules, as explained above). Executing EXPAR at a state Q yields a new state Q' where the values of \mathbf{a} and \mathbf{b} are swapped.

The general form of a parallel rule PAR is

par COND₁ ... COND_n endpar.

where COND₁, ..., COND_n are conditional rules. Executing PAR at a state A and variable assignment α will result in a simultaneous execution of the updates performed by COND₁, ..., COND_n. Formally, the update set of PAR is defined as

$$\text{PAR}_{A,\alpha} \stackrel{\text{def}}{=} \text{COND}_{1A,\alpha} \cup \dots \cup \text{COND}_{nA,\alpha}.$$

An update set Δ (such as PAR_{A,α}) then is applied to a state A by changing the function values according to the update molecules in Δ : for each update molecule $(\mathbf{f}, \bar{a}, v) \in \Delta$, the value of \mathbf{f} at argument \bar{a} is changed to v . The resulting state is denoted by $A \oplus \Delta$. Hence, the functions of $A \oplus \Delta$ are defined as

$$\mathbf{f}_{A \oplus \Delta}(\bar{a}) = \begin{cases} v & , \text{ for } (\mathbf{f}, \bar{a}, v) \in \Delta \\ \mathbf{f}_A(\bar{a}) & , \text{ otherwise} \end{cases}$$

for each n -ary function symbol \mathbf{f} and each n -ary argument tuple \bar{a} .

Note that $A \oplus \Delta$ is undefined in case Δ is *inconsistent*, i.e. Δ contains two molecules (\mathbf{f}, \bar{a}, v) and $(\mathbf{f}, \bar{a}, v')$ with $v \neq v'$. For example, executing the parallel rule EXPAR2

par f(x):=u f(y):=v endpar

at a state A with $\mathbf{x}_A = \mathbf{y}_A$ and $\mathbf{u}_A \neq \mathbf{v}_A$ yields an inconsistent update set. In that case EXPAR2 yields no next-state.

For technical convenience, we assume every conditional rule COND as a parallel rule **par COND endpar**.

4.4 Choice Rules

Choice rules allow nondeterministic choice of elements of the universe of a state. An example of a choice rule built over the signature Σ_Q is the rule EXCHOICE:

choose v do a:= v .

Executed at state Q , EXCHOICE chooses a value v from the universe of Q and assigns it to \mathbf{a} . Hence, EXCHOICE yields three different possible next-states, one for each of the values 1, 2, and 3. At a state A with an infinite universe U_A , EXCHOICE yields an infinite number of next-states, one for each element of U . Therefore, choice rules introduce *unbounded nondeterminism*.

A slightly advanced example of a choice rule is EXCHOICE2:

choose x, y **with** $v(x) \neq y$ **do** $v(x) := y$.

EXCHOICE2 assigns the nondeterministically chosen value y to the function symbol v at the nondeterministically chosen argument x . The additional condition $v(x) \neq v$ restricts the possible values for x and y . In this case, the condition ensures that the newly assigned value differs from the old one.

In general terms, a choice rule introduces quantified variables as known from first-order logic. The general form of a choice rule CHOICE is

choose x_1, \dots, x_n **with** ϕ **do** PAR

where x_1, \dots, x_n are variables, ϕ is a Boolean formula, and PAR is a parallel rule such that ϕ and PAR contain only variables from $\{x_1, \dots, x_n\}$ (i.e. all variables in CHOICE are bounded). For a given state A , CHOICE first nondeterministically chooses a variable assignment α such that ϕ is satisfied by A and α . Then PAR is executed by use of the variable assignment α . Consequently, CHOICE has the potential for infinitely many possible update sets at a state A . Formally, this set of possible update sets is defined as

$$\text{CHOICE}_A =_{\text{def}} \{ \text{PAR}_{A,\alpha} \mid \exists \alpha : \phi \text{ is satisfied by } A \text{ and } \alpha \}.$$

The semantics of a choice rule CHOICE built over a signature Σ is then defined in terms of a next-state relation $\rightarrow_{\text{CHOICE}}$: for two Σ -algebras A, A' holds $A \rightarrow_{\text{CHOICE}} A'$ iff there is a consistent update set $\Delta \in \text{CHOICE}_A$ such that $A' = A \oplus \Delta$.

4.5 Generalized Syntax and Semantics

The syntax of ASM rules presented above is rather restricted: a choice rule merely contains a collection of simultaneously executed assignment rules. Such a restriction definitely would hamper the practical application of ASM as a specification language.

To cope with this problem, the syntax and semantics of ASM rules may be extended to allow arbitrary nesting of conditional rules, parallel rules, and choice rules. In fact, ASM rules classically are defined to allow such arbitrary nesting [7, 5]. An example of such a nested ASM rule over signature Σ_Q is the following ASM rule NESTED:

```

par
  if (a=v(a)) then par
    choose  $x$  do a:= $x$ 
    choose  $y$  do b:= $y$ 
  endpar
  if ( $\neg$ a=v(a)) then b:=inc(b)
endpar.
```

In this paper we stick to the simple un-nested version of ASM rules in order to keep the technical details as low as possible. Nevertheless, this restriction

is not critical. The expressive power of ASM rules as presented in this paper does not increase by allowing arbitrary nesting: each nested ASM rule such as NESTED can be canonically transformed to an equivalent un-nested choice rule by shifting `choose`-statements towards the beginning of the rule (we skip the formal construction here). This corresponds to an analogy from first-order logic: every first-order formula built from \exists , \wedge and \vee can be transformed to its prenex normal form containing only a single occurrence of \exists .

According to their syntax, ASM rules merely seem to be yet another programming language. But in contrast to classical programs, an ASM rule RULE built over a signature Σ may be executed on *arbitrary* Σ -algebras. Therefore, as explained in the introduction, RULE may compute on arbitrary mathematical objects such as vectors and real numbers.

4.6 Nondeterministic ASMs

A nondeterministic ASM resembles a nondeterministic algorithm except for the next-state relation which is explicitly given by a choice rule. More precisely, a nondeterministic ASM M consists of

- a signature Σ_M ,
- a set of Σ_M -algebras \mathcal{S}_M , closed under isomorphism (the states of M),
- a set $\mathcal{J}_M \subseteq \mathcal{S}_M$, closed under isomorphism (the initial states of M),
- a choice rule CHOICE built over the signature Σ_M .

The next-state relation of M , denoted by \rightarrow_M , is the restriction of $\rightarrow_{\text{CHOICE}}$ to the states of M . Analogously to nondeterministic algorithms, a *run of M* is a sequence $A_0 A_1 A_2 \dots$ of states of M with $A_0 \in \mathcal{J}_M$ and $A_i \rightarrow_M A_{i+1}$ for all indices i .

5 The Equivalence Theorem

In this section we present the main result of this paper. The following theorem states that the class of nondeterministic algorithms (as introduced in Section 3) and the class of nondeterministic ASMs (as introduced in Section 4) are equivalent:

Theorem 1 (Equivalence Theorem). *Nondeterministic algorithms and nondeterministic ASMs describe the same set of transition systems.*

We present the proof in the next section.

Theorem 1 confirms that the notion of nondeterministic ASMs and the notion of nondeterministic algorithms may be used interchangeably. In particular, interesting properties of nondeterministic ASMs can be identified by examining nondeterministic algorithms. As an example, for each nondeterministic algorithm \mathcal{N} , reverting the next-state relation $\rightarrow_{\mathcal{N}}$ yields another nondeterministic algorithm. Hence, nondeterministic algorithms are reversible. This fact is proven by verifying the Axioms N1–N5 for the reverse of \mathcal{N} . This task is quite simple, as

the Axioms N1–N5 are highly symmetric with respect to the next-state relation $\rightarrow_{\mathcal{N}}$.

According to the Theorem 1, the following corollary follows immediately:

Corollary 1 (Reversibility). *For each nondeterministic ASM M there exists a nondeterministic ASM M^{-1} such that $\rightarrow_{(M^{-1})} = (\rightarrow_M)^{-1}$.*

Alternatively, Corollary 1 can also be proven without Theorem 1 by constructing for each ASM rule its reverse rule. For instance, the reverse of the rule “ $\mathbf{a} := \mathbf{next}(\mathbf{a})$ ” is the rule “ $\mathbf{choose } x \text{ with } \mathbf{a} = \mathbf{next}(x) \text{ do } \mathbf{a} := x$ ”. However, the proof based on Theorem 1 is considerably simpler, as there are no such syntactical constructions involved. This points out the profit of Theorem 1: the notion of nondeterministic algorithms allows to examine nondeterministic ASMs without any syntactic overhead.

6 Proof of The Equivalence Theorem

In this section we present the proof of Theorem 1. The proof divides into two parts: firstly, we show that every nondeterministic ASM represents a nondeterministic algorithm. Secondly, we show that every nondeterministic algorithm can be represented by a nondeterministic ASM.

The first part is fairly simple. Let \mathcal{M} be a nondeterministic ASM with CHOICE its choice rule. One only needs to verify that \mathcal{M} satisfies the axioms N1–N5: Axioms N1 and N2 are satisfied by the definition of nondeterministic ASMs. Axiom N3 and N4 are properties of the semantics of CHOICE that are easily verified. Axiom N5 holds due to the fact that in each step (A, A') of \mathcal{M} , CHOICE accesses and modifies only a bounded substate of A .

The second part of the proof requires considerably more effort. For the rest of this section, let \mathcal{N} be a nondeterministic algorithm, and let \mathcal{W} be a bounded-work witness for \mathcal{N} (see axiom N5). In the following we show that there exists a nondeterministic ASM \mathcal{M} that represents \mathcal{N} . Due to the significant difference between Gurevich’s bounded exploration axiom and our bounded work axiom, the proof employs a couple of new argumentations.

We start with a lemma presenting a close connection between completions of substeps and the \oplus -operator:

Lemma 1. *Let $(M, M') \in \mathcal{W}$ and let $A, A' \in \mathcal{S}_{\mathcal{N}}$. Then (A, A') is a completion of (M, M') iff $M \subseteq A$ and $A' = A \oplus M'$.*

Proof. (\Rightarrow) Let E be a substate disjoint from M and M' such that $(A, A') = (M \cup E, M' \cup E)$. Obviously $M \subseteq A$. Further show that $A \oplus M' = (A \setminus M) \cup M'$. As $A' = E \cup M' = (A \setminus M) \cup M'$, this implies $A' = A \oplus M'$. (\Leftarrow) For $E =_{\text{def}} A \setminus M$, show that E and M' are disjoint and that $(A, A') = (M \cup E, M' \cup E)$. \square

The next lemma states that parallel rules preserve isomorphisms between states. For a parallel rule PAR, let $\tau(\text{PAR}, A, \alpha) =_{\text{def}} A \oplus \text{PAR}_{A, \alpha}$ denote the next state computed by PAR at state A and variable assignment α .

Lemma 2. *Let PAR be a parallel rule over a signature Σ , let A, B be Σ -algebras with an isomorphism $i : A \rightarrow B$, and let α be a variable assignment with values in U_A . Then $i : \tau(\text{PAR}, A, \alpha) \rightarrow \tau(\text{PAR}, B, i \circ \alpha)$ also is an isomorphism.*

Proof. Follows by examining the semantics of PAR. \square

The following Lemma presents the main part of the proof: for each step (A, A') of \mathcal{N} , a choice rule CHOICE can be constructed such that (A, A') is a step of CHOICE, and all other steps of CHOICE are also steps of \mathcal{N} .

Lemma 3. *Let (A, A') be a step of \mathcal{N} . Then there is a choice rule CHOICE such that $A \rightarrow_{\text{CHOICE}} A'$, and $B \rightarrow_{\text{CHOICE}} B'$ implies $B \rightarrow_{\mathcal{N}} B'$ for all states B, B' of \mathcal{N} .*

Proof. By Axiom N5, there exists a substep $(M, M') \in \mathcal{W}$ such that (A, A') is a completion of (M, M') . We use (M, M') to construct CHOICE.

Let $V \subseteq U_A$ denote the elements of the universe of A occurring in M and M' . As the size of M and M' is bounded, V is finite. For each element $v \in V$, choose a unique variable x^v . Define the Boolean formulas ϕ and ψ by

$$\phi =_{\text{def}} \bigwedge_{v \neq w \in V} x^v \neq x^w, \quad \psi =_{\text{def}} \bigwedge_{(\mathbf{f}, [u_1, \dots, u_n], v) \in M} \mathbf{f}(x^{u_1}, \dots, x^{u_n}) = x^v.$$

Construct for each molecule $(\mathbf{f}, [u_1, \dots, u_n], v) \in M'$ the assignment rule $\mathbf{f}(x^{u_1}, \dots, x^{u_n}) := x^v$. Combine all of these assignment rules to a single parallel rule PAR. Let v_1, \dots, v_m be the elements in V . Define CHOICE as

$$\text{choose } x^{v_1}, \dots, x^{v_m} \text{ with } \phi \wedge \psi \text{ do PAR.}$$

According to axiom N5, the size of M and M' is bounded by a constant k . Consequently, the size of CHOICE also is bounded by a constant c .

We first show that $A \rightarrow_{\text{CHOICE}} A'$: let α be the variable assignment defined by $\alpha(x^v) = v$ for all $v \in V$. Then ϕ and ψ are satisfied by A and α . Further, by construction of PAR, we have $\text{PAR}_{A, \alpha} = M'$. Therefore, $A \rightarrow_{\text{CHOICE}} A \oplus M'$. By Lemma 1, $A \oplus M' = A'$.

Finally, let B, B' be states of \mathcal{N} such that $B \rightarrow_{\text{CHOICE}} B'$. We have to show that $B \rightarrow_{\mathcal{N}} B'$. As $B \rightarrow_{\text{CHOICE}} B'$, there is a variable assignment β such that ϕ and ψ are satisfied at B and β , and $B' = \tau(\text{PAR}, B, \beta)$.

Construct from B a isomorphic state C by bijectively replacing, for each $v \in V$, the element $\beta(x^v)$ by v , and by replacing every other element from U_B by a new element not contained in U_B . This construction is well-defined as ϕ is satisfied by B and β . Let $i : B \rightarrow C$ be the corresponding isomorphism. As ψ is satisfied by B and β , one can show that $M \subseteq C$. By construction of PAR, we have $\text{PAR}_{C, i \circ \beta} = M'$.

Let $C' =_{\text{def}} \tau(\text{PAR}, C, i \circ \beta)$. By Lemma 2, $i : B' \rightarrow C'$ is an isomorphism. As B, B' are states of \mathcal{N} , Axiom N4 implies that C, C' also are states of \mathcal{N} . As $\text{PAR}_{C, i \circ \beta} = M'$ and by the definition of $\tau(\text{PAR}, C, i \circ \beta)$, we conclude $C' = C \oplus M'$. As $M \subseteq C$, Lemma 1 implies that (C, C') is a completion of (M, M') . By Axiom N5, (C, C') is a step of \mathcal{N} . As $i : B \rightarrow C$ and $i : B' \rightarrow C'$ are isomorphisms, Axiom N4 implies that (B, B') also is a step of \mathcal{N} . \square

The last lemma states that a finite set of choice rules can be united to a single choice rule:

Lemma 4. *Let $\text{CHOICE}_1, \dots, \text{CHOICE}_n$ be choice rules. Then there exists a single choice rule UNION such that $\rightarrow_{\text{CHOICE}} = \rightarrow_{\text{CHOICE}_1} \cup \dots \cup \rightarrow_{\text{CHOICE}_n}$. CHOICE is the union of $\text{CHOICE}_1, \dots, \text{CHOICE}_n$.*

Proof. For the sake of simplicity, we present CHOICE in form of a nested non-deterministic ASM rule. However, as stated in Section 4, CHOICE may be transformed to an equivalent un-nested choice rule. The desired rule CHOICE is

```

choose  $x_0, \dots, x_n$ 
with  $\bigvee_{1 \leq i \leq n} (x_0 = x_i \wedge \bigwedge_{1 \leq j \leq n, j \neq i} x_0 \neq x_j)$  do par
  if  $(x_0 = x_1)$  then  $\text{CHOICE}_1$ 
  ...
  if  $(x_0 = x_n)$  then  $\text{CHOICE}_n$ 
endpar.

```

The choice condition ensures that the value of x_0 is equal to the value of one and only one x_i , $i = 1, \dots, n$. Hence, each variable assignment satisfying the choice condition satisfies exactly one of the guards $(x_0 = x_1), \dots, (x_0 = x_n)$.

A technical remark: as the above choice condition cannot be satisfied at states with singleton universes, the above construction works only for states whose universe contains at least two elements. However, this issue can be resolved by a rather technical extension of the above construction. Due to the lack of space and due to the practical irrelevance of singleton universes we skip this extension here. \square

The final proof combines the lemmata presented above:

Proof (of Theorem 1). For all steps (A, A') , derive a choice rule $\text{CHOICE}_{(A, A')}$ by applying Lemma 3. Let \mathcal{C} be the set of all of these choice rules. By construction (see proof of Lemma 3), the size of all choice rules in \mathcal{C} is bounded by a constant c . WLOG we may assume that the programs in \mathcal{C} contain only finitely many different variables. Hence, as \mathcal{C} contains only symbol sequences of bounded length over a finite alphabet, \mathcal{C} is finite.

By Lemma 4, let CHOICE be the union of all choice rules in \mathcal{C} . Then for all states A, A' of \mathcal{N} , we have $A \rightarrow_{\mathcal{N}} A'$ iff $A \rightarrow_{\text{CHOICE}} A'$: (\Rightarrow) Let $A \rightarrow_{\mathcal{N}} A'$. By Lemma 3, $A \rightarrow_{\text{CHOICE}_{(A, A')}} A'$. By Lemma 4, $A \rightarrow_{\text{CHOICE}} A'$. (\Leftarrow) Let $A \rightarrow_{\text{CHOICE}} A'$. By Lemma 4, there is a step (B, B') of \mathcal{N} with $A \rightarrow_{\text{CHOICE}_{(B, B')}} A'$. By Lemma 3, $A \rightarrow_{\mathcal{N}} A'$. \square

7 Conclusion

The theory of ASMs suggests a comprehensive and quite general approach to the notion of “algorithm”. The basic idea is to represent each state of an algorithm by an algebra. A number of variants of ASMs have been identified,

among them sequential, nondeterministic, parallel, and interactive versions. The deeper understanding of all variants of ASMs requires their characterization independently of any concrete syntax. This has been achieved for many of them, including sequential, parallel, and interactive versions.

In this paper we characterized the class of unbounded-nondeterministic ASMs. To this end we axiomatized the class of nondeterministic algorithms and showed that this class is equivalent to unbounded-nondeterministic ASMs. Surprisingly, the definition of nondeterministic algorithms turns out to be considerably simpler than the semantics of unbounded-nondeterministic ASMs. Due to this fact, we were able to prove the reversibility of unbounded-nondeterministic ASMs with nearly no effort. We intend to identify and to prove further interesting properties (such as linear speedup [8]) in a similar way.

Acknowledgement. We are grateful to the anonymous referees for their helpful comments.

References

1. Andreas Blass and Yuri Gurevich. Abstract State Machines Capture Parallel Algorithms. *ACM Trans. Comput. Logic*, 4(4):578–651, 2003.
2. Andreas Blass and Yuri Gurevich. Ordinary Interactive Small-Step Algorithms, parts I, II, III. *ACM Trans. Comput. Logic*, 2006. to appear.
3. Andreas Blass, Yuri Gurevich, Dean Rosenzweig, and Benjamin Rossman. General Interactive Small Step Algorithms. Technical Report MSR-TR-2005-113, Microsoft Research, Aug 2006.
4. Egon Börger. The Origins and the Development of the ASM Method for High Level System Design and Analysis. *Journal of Universal Computer Science*, 8(1):2–74, 2002.
5. Egon Börger and Robert Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.
6. Yuri Gurevich. A New Thesis. *Abstracts, American Mathematical Society*, page 317, Aug 1985.
7. Yuri Gurevich. Evolving Algebras 1993: Lipari Guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
8. Yuri Gurevich. Sequential Abstract State Machines Capture Sequential Algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, Jul 2000.
9. Yuri Gurevich and Tatiana Yavorskaya. On Bounded Exploration and Bounded Nondeterminism. Technical Report MSR-TR-2006-07, Microsoft Research, Jan 2006.
10. ITU-T. SDL Formal Semantics Definition. ITU-T Recommendation Z.100 Annex F, International Telecommunication Union, Nov 2000.
11. Wolfgang Reisig. On Gurevich’s Theorem on Sequential Algorithms. *Acta Informatica*, 39(5):273–305, 2003.
12. Robert Stärk, Joachim Schmid, and Egon Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer-Verlag, 2001.