

Some Things Algorithms Cannot Do

Dean Rosenzweig and Davor Runje

University of Zagreb

Abstract. Church–Turing thesis captured the notion of a computable function by defining an envelope containing all functions that could be computed by a mechanical procedure. The intended use of the thesis was negative — it provided a theoretical framework for undecidability results. The ASM thesis captured the notion of algorithm, rather than the function it computes. The intended use of the thesis was positive application — it has been most successful in specification of algorithms.

We raise the question can the thesis be used for negative purposes as well. More precisely, we attempt to give a meaningful interpretation to negative abstract results in the context of an ASM model of abstract cryptography. In this paper we develop a theoretical framework for establishing negative result in the general behavioral theory of algorithms enabling such negative applications of the ASM thesis. The algorithms studied are small-step algorithms, non-interactive and ordinary interactive, possibly importing from a background.

Introduction

Church–Turing thesis captured the notion of computable function — a function computed by an algorithm. The intended use of the thesis was negative; Alonzo Church [8] in 1936, and Alan Turing [16] few months later, gave two independent but equivalent definitions of an envelope containing all functions that could possibly be computed by a mechanical procedure. If a function is not contained in the envelope, then it is certainly not computable by any practical means. On the other hand, the sole fact that a function is contained in this envelope does not provide us with any guarantee that a practical procedure for computing the function exists. It is hardly a surprise that the most spectacular applications of the thesis – a long stream of undecidability results – were negative.

A new, ‘behavioral’ theory of algorithms has been recently developed in a series of papers by Yuri Gurevich, Andreas Blass, Benjamin Rossman and the first author, motivated initially by the goal of establishing the Abstract State Machine (ASM) thesis. The goal of the ASM thesis was to capture the notion of algorithm itself, rather than the function it computes, at its intended level of abstraction. The intended use of the thesis was positive, and it has been used highly successfully in positive applications such as specification and verification. A wide variety of algorithms were captured at their natural abstraction levels by appropriate ASMs. No specific algorithm or system presenting a conceptual challenge for the methodology is found until today; the challenge of modeling a system is inherited from the system modeled, and not by the methodology.

The ASM methodology was also very successful in relating models of the same algorithm at different level of abstraction. A typical specification of a system starts with a high level description that is succeedingly refined to include more details at lower levels of abstraction. See [7] for a long list of successful applications of the refinement methodology in the context of ASMs.

Can the ASM thesis be used for negative results as well? To be more precise, can we give a meaningful interpretation to the following type of a negative result: if there is no algorithm, at some fixed level of abstraction, that can perform a given action, what does it really mean? Sometimes, it has no meaning other than that we are working at a wrong level of abstraction.

But sometimes it can be given a meaningful interpretation and used for practical purposes. One example of such practical purpose is the field of abstract cryptography. In a widely accepted computational model of cryptography, agents running probabilistic polynomial time (PPT) Turing machines are exchanging strings over an open network, and the security of such systems is expressed in terms of probabilities of an attack mounted by a PPT agent controlling the network. To remedy the intrinsic difficulty of reasoning about such systems, an abstract model of cryptography abstracts from concrete strings and PPT algorithms, and replaces them with abstract messages and abstract algorithms working over such messages. The messages are typically represented with terms of some vocabulary including constructs for representing encryptions, keys and pairs. The capabilities of an abstract agent are then defined in terms of deducibility of an abstract message from a given set of abstract messages. Such rules are simple and intuitive, e.g. a typical rule looks like:

- *given* a key K and an encryption e *created* with a key K and subject m , an abstract agent can *deduce* m ,

where *given*, *created* and *deduce* are interpreted differently in different models, although they share a common intuitive meaning. Notice that there are no considerations of probabilities and capabilities of PPT Turing machines in the abstract model, making the model easier to reason about.

There is vast amount of work in the field of abstract cryptography; to put it mildly, every self-respecting formalism has been used for specification and verification of cryptographic protocols. Some models are more natural than others in the context of the underlying formalism, but they are really not as different as they appear to be — up to notation and idiosyncrasies, they are all the same. This should hardly be surprising; after all, they are nothing but abstractions of the same computational model.

We have also developed an abstract model of cryptography for which we claim it is the most natural one: capabilities of abstract agents are exactly the capabilities of sequential algorithms, in the sense of [12, 3–6], at a fixed abstraction level. In the companion paper [13], we present our ASM model of cryptography with algorithms working over states of vocabulary containing symbols for cryptographic operations such as encryption and decryption. Unlike other approaches, our model can have two distinct interpretations: a computational and an abstract interpretation, turning the same ASM model into models of computational and abstract cryptography. This unique feature allows us to easily relate two models and prove soundness and completeness theorems; i.e. no abstract algorithm can perform a certain action in the abstract model if and only if there is no PPT algorithm that can perform the same, but computationally interpreted, action in the computational model. The intrinsic ease of moving between different abstraction levels in the ASM methodology was the key advantage over other formalisms in proving the result. Due to space constraints, we must refer a sceptical reader to [13, 14] for justifications of the above claims.

Before being able to develop our model of cryptography, we had to develop a theoretical framework for negative applications of the behavioral theory of algorithms. This is done in this paper. Although motivated by abstract cryptography, we believe that the framework has many potential applications. The algorithms we study here are non-interactive [12], and ordinary interactive [4–6], small-step algorithms. We also investigate behavior of algorithms importing from a background in the sense of [2]. This choice of algorithms of interest was purely pragmatic — this is exactly what we needed for our model of abstract cryptography.

In order to make the paper reasonably self-contained, we also list several results which are not new, and which can be found scattered, sometimes inlined in proofs, sometimes without an explicit statement, in the behavioral theory literature. We attempt to attribute such results properly. All relevant notions and definitions taken from the literature were

also included for a reference, but some familiarity with [12, 4–6, 2] is expected from a reader.

The paper does not include too many examples to help the reader understand the intended use of the presented theory. Our excuse is that the intended use can be best understood by reading the companion paper [13]. Due to space constraint, we were unable to include a short overview here.

We thank Andreas Blass, Matko Botinčan and Yuri Gurevich for very helpful comments on earlier versions of the paper.

1 Non-Interactive Small-Step Algorithms

1.1 Preliminaries

We take over many notions, notations and conventions on vocabularies, structures and sequential algorithms from [12] without further ado. In particular, we assume the following:

- all structures we consider are purely functional (i.e. algebras);
- all vocabularies have distinguished nullary symbols `true`, `false` and `undef`, with the interpretation of `true` distinct from interpretations of `false` and `undef` in all structures considered;
- all vocabularies have the binary function symbol `=`, interpreted as identity in all structures, as well as the usual Boolean connectives with their usual interpretations. If one of the arguments of a Boolean connective is not Boolean, the connective takes the default value of `false`.

Symbols `true`, `false`, `undef`, `=` and connectives are called *logical constants*.

Ground terms of a vocabulary \mathcal{Y} are defined inductively in the usual way. All terms in this section are assumed to be ground. We will write $t' \sqsubseteq t$ if t' is a subterm of t .

The definitions of coincidence and similarity of structures were introduced in [12]. We list them here again and fix the notation used in the rest of the paper:

Definition 1. Let \mathcal{Y} be a vocabulary and T a set of \mathcal{Y} -terms. \mathcal{Y} -structures X and Y are said to *coincide over T* , denoted with $X =_T Y$, if every term in T has the same value in X and Y .

A structure X induces an equivalence relation E_X on T : $(t_1, t_2) \in E_X$ if and only if $Val(t_1, X) = Val(t_2, X)$.

Definition 2. Let \mathcal{Y} be a vocabulary and T a set of \mathcal{Y} -terms. \mathcal{Y} -structures X and Y are *T -similar*, written as $X \sim_T Y$, if they induce the same equivalence relation over T .

Both relations are equivalence relations over \mathcal{Y} -structures for any choice of T . For any fixed set of terms T , coincidence is contained in similarity: if $X =_T Y$, then $X \sim_T Y$. Isomorphic structures are also similar: if $X \cong Y$, then $X \sim_T Y$.

When T is the set of *all* \mathcal{Y} -terms, we suppress it, and speak of coincident and similar structures.

The following theorem reveals the connection between the equivalence relations on structures just mentioned. It is implicit in the proof of one of the key lemmas of [12]—it is actually proved there, although not explicitly stated.

Proposition 1 (Factorization). *Let X and Y be structures of a vocabulary \mathcal{Y} , T a set of \mathcal{Y} -terms. Then X, Y are T -similar if and only if there is a structure Z isomorphic to Y which coincides with X over T .*

It follows immediately that similarity is the joint transitive closure of isomorphism and coincidence:

Corollary 1. *Let T be a set of \mathcal{Y} -terms. Similarity \sim_T is the smallest transitive relation over \mathcal{Y} -structures containing both coincidence $=_T$ and isomorphism \cong .*

1.2 Postulates

Gurevich in [12] defines a *sequential algorithm* as an object A satisfying a few simple postulates. We use an alternative formulation of the same principles from [4] and list them below for reference. We will make no attempt to justify the postulates or defend the thesis here — a sceptical reader will find the speculative argumentation in favor of this definition in [12].

Postulate 1 (State) *Every algorithm A determines*

- a nonempty collection $\mathcal{S}(A)$, called states of A ;
- a nonempty collection $\mathcal{I}(A) \subseteq \mathcal{S}(A)$, called initial states; and
- a finite vocabulary \mathcal{Y} such that every $X \in \mathcal{S}(A)$ is an \mathcal{Y} -structure.

If, for some state X , f is an n -ary function symbol of its vocabulary and $a_1, \dots, a_n \in X$, then $f, (a_1, \dots, a_n)$ is a *location* of X and $l_X = f_X(a_1, \dots, a_n)$ is its value in X .

The difference of two structures X, Y of the same vocabulary and the same base set can be explicitly represented as the *update set*

$$\begin{aligned} Y - X &= \{(l, a_0) \mid l_Y = a_0 \neq l_X\} \\ &= \{(f, (a_1, \dots, a_n), a_0) \mid f_Y(a_1, \dots, a_n) = a_0 \neq f_X(a_1, \dots, a_n)\}. \end{aligned}$$

Given an update set Δ and a structure X , we can determine a unique structure Y such that $Y - X = \Delta$ as follows:

- Y has the same base set as X ,
- $f_Y(a_1, \dots, a_n) = \begin{cases} b & \text{if } (f, (a_1, \dots, a_n), b) \in \Delta \\ f_X(a_1, \dots, a_n) & \text{otherwise} \end{cases}$

We will write $Y = X + \Delta$ when $\Delta = Y - X$. We say that an update set Δ is *contradictory* if $(f, (a_1, \dots, a_n), b_1) \in \Delta$ and $(f, (a_1, \dots, a_n), b_2) \in \Delta$ for some $b_1 \neq b_2$.

Postulate 2 (Updates) *For any state X the algorithm provides an update set $\Delta_A(X)$. If the update set is contradictory, the algorithm fails; otherwise it produces the next state $\tau_A(X) = X + \Delta_A(X)$.*

States of an algorithm are *abstract*, in the sense that everything relevant in a state must be represented by vocabulary: if your algorithm can distinguish red integers from green integers, then it is not about integers.

Postulate 3 (Isomorphism) – *Any structure isomorphic to a state is a state.*

- *Any structure isomorphic to an initial state is an initial state.*
- *If $i : X \cong Y$ is an isomorphism of states, then $i[\Delta_A(X)] = \Delta_A(Y)$.*

An algorithm performs a bounded amount of work in each state using its state as an oracle: it explores a part of its state by evaluating some terms, compares if some terms have the same value in the state or not, and based on such comparisons evaluates some other terms or completes its step and computes a new state. The intuition that evaluation of a term is a good measurement of amount of work an algorithm is performing in a state is expressed in the following postulate:

Postulate 4 (Bounded Exploration) *There is a finite set of terms T such that $\Delta_A(X) = \Delta_A(Y)$ whenever X and Y coincide over T .*

Such a set of terms is a *bounded exploration witness* for A . Notice that a bounded exploration witness is not uniquely determined, e.g. any finite superset of a witness is also a witness. Whenever we refer to a bounded exploration witness T , we assume that for a given algorithm we have chosen an arbitrary but fixed set of terms satisfying the postulate. We shall also call terms in T *critical* or *observable*. We do not assume that a bounded exploration witness is closed under subterms.

Since many tend to understand a sequential algorithm simply as a non-distributed algorithm, [4] suggest a confusion-preventing shift in terminology: an object satisfying the above postulates could be aptly called a *small-step algorithm*. We will adhere to that here.

An element $a \in X$ is *critical* at X if it is the value of a critical term, given an algorithm A and its fixed bounded exploration witness T . For reference, we list the following lemma, proved in [12]:

Lemma 1. *If $(f, (a_1, \dots, a_n), a_0) \in \Delta_A(X)$, then every a_i is critical at X .*

By the above lemma and Bounded Exploration postulate, the update set of a small-step algorithm is bounded in all states.

1.3 Next Value

The main topic of this section is development of an answer to the following question: given an algorithm A and an arbitrary term t , is there a term t' such that $Val(t, \tau_A(X)) = Val(t', X)$ for every state X ?

For a fixed algorithm A with a bounded exploration witness T and a fixed term t , we will show that there is a finite set of terms T^t partitioning the set of states \mathcal{S}_A into a finite set of T^t -similar equivalence classes $\{[X_1]_{\sim_{T^t}}, \dots, [X_n]_{\sim_{T^t}}\}$, where $[X]_{\sim_{T^t}} = \{X' \mid X \sim_{T^t} X'\}$, and we will associate a fixed term $\text{Next}_X^A(t)$ from T^t to each $[X]_{\sim_{T^t}}$, such that for every state $Y \in [X]_{\sim_{T^t}}$ we have:

$$Val(t, \tau_A(Y)) = Val(\text{Next}_X^A(t), Y).$$

Since T^t is finite, this means that an algorithm can determine the equivalence class of the current state and determine the value of a term in its next state.

We will proceed by showing that a step of a small-step algorithm preserves coincidence and similarity over the set of all terms, and that an element denotable by a term in the successor state of a state X was denotable by a (possibly different) term in X . We will also show how this can be used to derive some known results like Linear Speedup.

Fix vocabulary \mathcal{Y} . Let T be a set of terms and t a term. Then T^t is the smallest set of terms containing t and T closed under subterms and substitutions of subterms of t with terms of T , i.e. T^t is defined with the following induction:

- $\{t\} \cup T \subseteq T^t$;
- if $t' \sqsubseteq t'' \in T^t$, then $t' \in T^t$; and
- if $t' \sqsubseteq t$ and $t'' \in T$, then $t[t''/t'] \in T^t$.

An alternative definition of T^t can be made by the following induction on the structure of the term t :

- for $t = f$, we have $T^t = T \cup \{f\}$;
- for $t = f(t_1, \dots, t_n)$, we have $T^t = T \cup \{f(t'_1, \dots, t'_n) \mid t'_i \in T^{t_i}\} \cup \{T^{t_i} \mid i = 1, \dots, n\}$.

Obviously, if T is finite, T^t is finite as well. It is also easy to see that if $t_1 \sqsubseteq t_2$ and $T_1 \subseteq T_2$, then $T_1^{t_1} \subseteq T_2^{t_2}$.

Lemma 2. *Let A be a small-step algorithm, X its state, T its exploration witnesses, and t a term of its vocabulary. Then there is a term $\text{Next}_X^A(t) \in T^t$ such that*

- $Val(\text{Next}_X^A(t), X) = Val(t, \tau_A(X))$, moreover,
- for every state Y coinciding with X over T^t , $Val(t, \tau_A(Y)) = Val(t, \tau_A(X))$.

Proof. We construct the term $\text{Next}_X^A(t)$ and prove the statements by induction on the structure of t .

We proceed with a step of induction. Suppose that $t = f(t_1, \dots, t_n)$ with $Val(t_i, \tau_A(X)) = a_i$. From the induction hypothesis we have:

- there is a term $\text{Next}_X^A(t_i) \in T^{t_i}$ such that $Val(\text{Next}_X^A(t_i), X) = a_i$, and
- whenever $Y =_{T^{t_i}} X$ then also $Val(t_i, \tau_A(Y)) = a_i$.

Since $t_i \sqsubseteq t$, we have $T^{t_i} \subseteq T^t$ for every t_i . By the induction hypothesis we have:

$$\begin{aligned} Val(t, \tau_A(Y)) &= f_{\tau_A(Y)}(Val(t_1, \tau_A(Y)), \dots, Val(t_n, \tau_A(Y))) \\ &= f_{\tau_A(Y)}(a_1, \dots, a_n) \end{aligned} \quad (1)$$

We have the following two cases:

1. Assume $(f, (a_1, \dots, a_n), a_0) \in \Delta_A(X)$ for some a_0 . By Lemma 1, a_0 is critical in X and there is a term $\text{Next}_X^A(t) \in T \subseteq T^t$ such that $Val(\text{Next}_X^A(t), X) = a_0 = Val(t, \tau_A(X))$. Suppose $Y =_{T^t} X$. Thus $\Delta_A(Y) = \Delta_A(X)$ and $(f, (a_1, \dots, a_n), a_0) \in \Delta_A(Y)$. We have

$$\begin{aligned} Val(t, \tau_A(Y)) &= f_{\tau_A(Y)}(a_1, \dots, a_n) && \text{from (1)} \\ &= a_0 && \text{from } (f, (a_1, \dots, a_n), a_0) \in \Delta_A(Y) \\ &= Val(t, \tau_A(X)) \end{aligned}$$

2. Otherwise, $(f, (a_1, \dots, a_n), a_0) \notin \Delta_A(X)$. Then $\text{Next}_X^A(t) = f(\text{Next}_X^A(t_1), \dots, \text{Next}_X^A(t_n)) \in T^t$, and

$$Val(t, \tau_A(X)) = f_{\tau_A(X)}(a_1, \dots, a_n) = f_X(a_1, \dots, a_n) = Val(\text{Next}_X^A(t), X)$$

Suppose $Y =_{T^t} X$. Then $(f, (a_1, \dots, a_n), a_0) \notin \Delta_A(Y)$ for any a_0 . Thus

$$\begin{aligned} Val(t, \tau_A(Y)) &= f_{\tau_A(Y)}(a_1, \dots, a_n) && \text{(from (1))} \\ &= f_Y(a_1, \dots, a_n) && \text{(from } (f, (a_1, \dots, a_n), a_0) \notin \Delta_A(Y)) \\ &= f_Y(Val(\text{Next}_X^A(t_1), X), \dots, Val(\text{Next}_X^A(t_n), X)) && \text{(by def. of } a_i) \\ &= f_Y(Val(\text{Next}_X^A(t_1), Y), \dots, Val(\text{Next}_X^A(t_n), Y)) && \text{(by } X =_{T^{t_i}} Y) \\ &= Val(f(\text{Next}_X^A(t_1), \dots, \text{Next}_X^A(t_n)), Y) = Val(\text{Next}_X^A(t), Y) \\ &= Val(\text{Next}_X^A(t), X) && \text{(from } X =_{T_i} Y) \\ &= Val(t, \tau_A(X)) \end{aligned}$$

The proof of the basis of induction (when $t = f$ for some nullary symbol f) is similar to the proof above. \square

Corollary 2 (Preserving Coincidence). *Let A be a small-step algorithm and X and Y coincident states (in all terms). Then $\tau_A(X)$ and $\tau_A(Y)$ coincide.*

Theorem 1 (Next Value). *Let A be a small-step algorithm and T its bounded exploration witness. If states X and Y are T^t -similar, then $Val(\text{Next}_X^A(t), Y) = Val(t, \tau_A(Y))$.*

Proof. By Theorem 1, there is a structure Z isomorphic to Y that coincide with X over T^t . By Abstract State postulate, Z is a state and $i : Z \cong Y$ is an isomorphism from $\tau_A(Z)$ onto $\tau_A(Y)$. Hence $Val(\text{Next}_X^A(t), Y) = i(Val(\text{Next}_X^A(t), Z))$ and $Val(t, \tau_A(Y) = i(Val(t, \tau_A(Z)))$. By Lemma 2, $Val(\text{Next}_X^A(t), Z) = Val(t, \tau_A(Z))$. But then we can conclude that $i(Val(\text{Next}_X^A(t), Z)) = i(Val(t, \tau_A(Z)))$ and finally

$$Val(\text{Next}_X^A(t), Y) = i(Val(\text{Next}_X^A(t), Z)) = i(Val(t, \tau_A(Z))) = Val(t, \tau_A(Y)). \quad \square$$

Corollary 3 (Preserving Similarity). *Let A be a small-step algorithm, and X and Y similar states. Then $\tau_A(X)$ and $\tau_A(Y)$ are similar.*

The following statement, quoted in [12] and proved for interactive algorithms in [6] (also proved by syntactic means in different places for different kinds of textual programs), states that whatever a small-step algorithm can do in two steps, could be done in one step by another small-step algorithm. By induction the same holds for any finite number of steps — the small steps can be enlarged by any fixed factor. We obtain it as a simple consequence of Next Value.

Proposition 2 (Linear Speedup). *Let A be a small-step algorithm, with associated $\mathcal{S}(A), \mathcal{I}(A)$ and τ_A . Then there is a small-step algorithm B , such that $\mathcal{S}(B) = \mathcal{S}(A)$, $\mathcal{I}(B) = \mathcal{I}(A)$, and $\tau_B(X) = \tau_A(\tau_A(X))$ for all $X \in \mathcal{S}(B)$.*

Proof. It suffices to demonstrate a bounded exploration witness for B . Let T be a bounded exploration witness for A , and X and Y be states of A . We have

$$\Delta_B(X) = \tau_A(\tau_A(X)) - X = \Delta_A(\tau_A(X)) \cup \{(l, a) \mid (l, a) \in \Delta_A(X) \wedge (l, a') \notin \Delta_A(\tau_A(X))\}$$

If X and Y coincide over T , we have $\Delta_A(X) = \Delta_A(Y)$. If they also coincide over a finite set $T^T = \bigcup\{T^t \mid t \in T\}$ extending T , then, by Next Value theorem, $\tau_A(X)$ coincides with $\tau_A(Y)$ over T . Hence, $\Delta_A(\tau_A(X)) = \Delta_A(\tau_A(Y))$ and $\Delta_B(X) = \Delta_B(Y)$. Thus T^T is a bounded exploration witness for B . \square

The similarity relation over a finite set of terms T^t partitions \mathcal{Y} -structures into finitely many equivalence classes — there is a finite set of structures $\{X_1, \dots, X_n\}$ such that every structure is T^t -similar to some X_i . For each X_i there is a Boolean term $\varphi(X_i, T^t)$ such that $\varphi(X_i, T^t)$ holds in Y if and only if Y is T^t -similar to X_i :

$$\varphi(X, T^t) = \bigwedge_{t_i, t_j \in T^t} (t_i \square_{i,j}^X t_j), \quad \text{where } \square_{i,j}^X = \begin{cases} = & \text{if } \text{Val}(t_i, X) = \text{Val}(t_j, X) \\ \neq & \text{otherwise} \end{cases}.$$

This was the crucial observation behind the proof of the sequential thesis [12] — it allowed uniformization of local update sets into a finite program. It also allows us to uniformize the $\text{Next}_X^A(t)$ construction into a single term for all states, given the following additional assumptions on vocabulary and states: in the rest of the section, we assume that the vocabulary of an algorithm contains a ternary symbol **if-then-else** interpreted in every state X as follows:

$$\text{if-then-else}_X(a_1, a_2, a_3) = \begin{cases} a_2 & \text{if } a_1 = \text{true}_X \\ a_3 & \text{otherwise} \end{cases}$$

We will typically write **if t_1 then t_2 else t_3** instead of **if-then-else**(t_1, t_2, t_3).

Theorem 2. Let A be a small-step algorithm, \mathcal{Y} its vocabulary, and t an \mathcal{Y} -term. Then there is a \mathcal{Y} -term $\text{Next}^A(t)$ such that $\text{Val}(\text{Next}^A(t), X) = \text{Val}(t, \tau_A(X))$ for every state X .

Different versions of the next-value construction, restricted to Boolean terms (logical formulæ, for which the **if-then-else** construct is definable), and proved over textual programs, have been around in the literature in the form of a ‘next-state’ modality [9, 1, 15].

1.4 Indistinguishability, Accessibility and Reachability

States of an algorithm at a fixed abstraction level can be viewed as (first-order) structures of fixed vocabulary. What is the natural notion of equivalence of such states? One might

argue it is isomorphism, claiming that everything relevant for algorithm execution in a state is expressed in terms of a class of structures isomorphic to it. After all, this is the intuition behind the postulates.

We will show that isomorphism is too fine-grained for some applications, not relating states that are (in any practical way) behaviorally indistinguishable by algorithms. Following the rich tradition of seeing the objects indistinguishable by a class of algorithms as equal, we will introduce the dynamic notion of indistinguishability by algorithms and show its equivalence with the static notion of similarity of structures. This equivalence will survive generalization to the case of algorithms that interact with the environment within a step in the next section.

The intuition behind the definition is that an algorithm can distinguish state X from state Y if it can determine in which of them it has executed a step. What does *to determine* mean here? Taking a behavioral view, we can require an algorithm to take different actions depending on whether it is in X or in Y , say by writing \mathbf{true}_X into a specific location if it is in X , but not if it is in Y .

Definition 3 (Indistinguishability). Let A be a small-step algorithm of the vocabulary \mathcal{Y} , whose states include X and Y . We say that A *distinguishes* X from Y if there is a \mathcal{Y} -term t taking the value \mathbf{true}_X in $\tau_A(X)$, and not taking the value \mathbf{true}_Y in $\tau_A(Y)$. Structures X and Y of the same vocabulary are *indistinguishable* by small-step algorithms if no such algorithm can distinguish them.

This is at first glance weaker than requiring of t to take the value of \mathbf{false}_Y in $\tau_A(Y)$, but only at first glance: if t satisfies our requirement, then the term $t = \mathbf{true}$ will satisfy the seemingly stronger requirement. The wording of Indistinguishability definition has been chosen so as to work smoothly also in an interactive situation, where terms can have no value. In spite of the asymmetric wording, it is easy to verify the following

Corollary 4. *Indistinguishability is an equivalence relation on structures of the same vocabulary.*

The dynamic notion of indistinguishability coincides with the static notion of similarity:

Theorem 3. *Structures X and Y of the same vocabulary \mathcal{Y} are indistinguishable by small-step algorithms if and only if they are similar.*

Proof. Suppose that X and Y are not similar. Then there are \mathcal{Y} -terms t_1 and t_2 having the same value in X and different values in Y . But then a do-nothing algorithm distinguishes them by term $t_1 = t_2$.

Now suppose that X and Y are similar and distinguishable by a term t taking the value \mathbf{true}_X in $\tau_A(X)$ and not \mathbf{true}_Y in $\tau_A(Y)$. Then $\tau_A(X)$ and $\tau_A(Y)$ are not similar, which is a contradiction by Corollary 3. \square

By Corollary 3, similarity is equivalent to indistinguishability in any number of steps. An element of a structure can be, in the small-step case, accessible to an algorithm only if it is the value of some term.

Definition 4 (Accessibility). An element a is *accessible* in a structure X of a vocabulary \mathcal{Y} if there is a \mathcal{Y} -term t such that $\text{Val}(t, X) = a$.

Remark 1. The reader familiar with logic should have in mind that we are speaking about indistinguishability *by algorithms*, and not about indistinguishability *by logic*: similar (indistinguishable) structures need not be elementarily equivalent. In all our examples of indistinguishable structures below it will be easy to find simple quantified sentences which distinguish them. But small-step algorithms are typically not capable of evaluating quantifiers over their states, unless such a capability is explicitly built in—if an algorithm explored states of unbounded size, the capability to evaluate quantifiers would contradict Bounded Work.

A straightforward consequence of Next Value is

Corollary 5. *Let A be a small-step algorithm and x an element of its state X . If x is accessible in $\tau_A(X)$, then it is accessible in X .*

Thus in a sense algorithms cannot learn anything by execution: they cannot learn how to make finer distinctions, and they cannot learn how to access more elements (but they can lose both kinds of knowledge). The only possibility of learning open to algorithms seems to be interaction with the environment, but this is the subject of subsequent sections. What states can algorithms reach?

Definition 5 (Reachability). A structure Y is *reachable* from a structure X of the same vocabulary and same base set by small-step algorithms if there is a small-step algorithm A such that $X, Y \in \mathcal{S}(A)$ and $Y = \tau_A(X)$.

By Linear Speedup, reachability in $\leq n$ steps is the same as reachability in one step, for any n . The notion of accessibility suffices to analyze reachability:

Theorem 4. *Let X, Y be structures of a vocabulary \mathcal{Y} with the same base set. Then Y is reachable from X by small-step algorithms if and only if*

- $Y - X$ is finite,
- all function symbols occurring in $Y - X$ are dynamic in \mathcal{Y} , and
- all objects in the common base set, occurring in $Y - X$, are accessible in X .

2 Ordinary Interactive Small-Step Algorithms

In [4–6] the theory was extended to algorithms interacting with the environment, also within a step. We refer the reader to [4] for full explication and motivation—it will have to suffice here to say that the essential goal of behavioral theory, that of capturing algorithms at arbitrary levels of abstraction, cannot be smoothly achieved if interaction with the environment is confined to happen only between the steps of the algorithm. The “step” is in the eye of beholder: what is say from socket abstraction seen as a single act of sending a byte-array may on a lower layer of TCP/IP look as a sequence of steps of sending and resending individual packets until an acknowledgment for each individual packet has arrived. In order to sail smoothly between levels of abstraction, we need the freedom to view several lower-level steps as compressed into one higher-level step when this is natural, even if the lower-level steps are punctured with external interaction.

The syntax of interaction can be, without loss in generality, given by a finite number of *query-templates* $\hat{f} \#1 \dots \#n$, each coming with a fixed arity. If b_1, \dots, b_n are elements of a state X , a *potential query* $\hat{f}[b_1, \dots, b_n]$ is obtained by instantiating the template positions $\#i$ by b_i ¹. The environment behavior can be, for the class of ordinary interactive algorithms, represented by an *answer function* over X : a partial function mapping potential queries to elements of X .

All algorithms in the rest of this paper are small-step ordinary interactive algorithms in this sense—in the sequel, we shall skip all these adjectives except possibly for “interactive”, to stress the difference with respect to algorithms of the previous section.

The interactive behavior of an algorithm is abstractly represented by a *causality relation*, between finite answer functions and potential queries. We have the following additional postulate:

Postulate 5 (Interaction) *The algorithm determines, for each state X , a causality relation \vdash_X between finite answer functions and potential queries.*

¹ The sole purpose of the $\hat{f}[b_1, \dots, b_n]$ notation is to be optically distinct from notation for function value $f(b_1, \dots, b_n)$ when $f \in \mathcal{Y}$.

The intuition of $\alpha \vdash_X q$ is: if the environment, in state X , behaves according to α , then the algorithm will issue q . A *context* for an algorithm is a minimal answer function that saturates the algorithm, in the sense that it would issue no more queries: α is a context if it is a minimal answer function with the following property: if $\beta \vdash_X q$ for some $\beta \subseteq \alpha$, then $q \in \text{Dom}(\alpha)$.

The Updates Postulate is modified by

- associating either failure or an update set Δ_A^+ to pairs X, α , where α is a context over X ;
- the update set $\Delta_A^+(X, \alpha)$ may also include trivial updates — in an interactive multi-algorithm situation trivial updates may express conflict with another component.

The Isomorphism Postulate is extended to preservation of causality, failure and updates, where $i : X \cong Y$ is extended to “extended states” X, α as $i : X, \alpha \cong Y, i \circ \alpha \circ i^{-1}$.

We can access elements of “extended states” X, α by “extended terms”, allowing also query-templates in the formation rules (the extended terms correspond to “e-tags” of [6]). Given vocabularies \mathcal{T} of function symbols, and E of query-templates disjoint from \mathcal{T} , we can (partially) evaluate extended terms as

$$\begin{aligned} \text{Val}(f(t_1, \dots, t_n), X, \alpha) &= f_X(\text{Val}(t_1, X, \alpha), \dots, \text{Val}(t_n, X, \alpha)) && \text{if } f \in \mathcal{T} \\ \text{Val}(\hat{f}(t_1, \dots, t_n), X, \alpha) &= \alpha(\hat{f}[\text{Val}(t_1, X, \alpha), \dots, \text{Val}(t_n, X, \alpha)]) && \text{if } f \in E \end{aligned}$$

if $\text{Val}(t_i, X, \alpha)$ are all defined, and $\hat{f}[\text{Val}(t_1, X, \alpha), \dots, \text{Val}(t_n, X, \alpha)] \in \text{Dom}(\alpha)$ in the latter case.

Thus the value of an extended term containing query templates can be undefined at X, α , which is different than being defined with the value undef_X . We shall in the sequel use equality of partially defined expressions in the usual Kleene-sense: either both sides are undefined, or they are both defined and equal.

Remark 2 (Kleene Equality). This means that we lose something of the tight correspondence that the meta-statement $\text{Val}(t_1, X) = \text{Val}(t_2, X)$ and the Boolean term $t_1 = t_2$ had in the noninteractive case: the former was true if and only if the latter had the (same) value (as) True. Now if say $\text{Val}(t_1, X, \alpha)$ is undefined, then also $\text{Val}(t_1 = t_2, X, \alpha)$ will be undefined, and the meta-statement $\text{Val}(t_1, X, \alpha) = \text{Val}(t_2, X, \alpha)$ will be either true or false, depending on whether $\text{Val}(t_2, X, \alpha)$ is also undefined. The reader should be aware of this when parsing the meta-statements about coincidence and similarity below.

The Bounded Work Postulate can be (equivalently to the formulation of [4–6] formulated as before, applying to extended terms, see [6] for extended discussion of “e-tags”).

The definition of critical elements must take into account answer functions attached to the state [4, Definition 3.5]: if α is an answer function for a state X , an element of X is *critical* for α if it is the value of some term in a bounded exploration witness T .

All elements in the update set for a given context are critical [4, Proposition 5.24]:

Lemma 3. *Let X be a state and α a context for X . For any update $\langle f, (a_1, \dots, a_n), a_0 \rangle \in \Delta_A^+(X, \alpha)$, every a_i is critical for α .*

2.1 Coincidence and Similarity

In this subsection, we will extend the notions of coincidence and similarity of extended terms to structures equipped with answer functions.

Definition 6 (Coincidence and Similarity). Let X, Y be \mathcal{T} -structures, α, β answer functions for X, Y , respectively, and T a set of extended terms. We say that

- X, α and Y, β coincide over T , and write $X, \alpha =_T Y, \beta$, if $\text{Val}(t, X, \alpha) = \text{Val}(t, Y, \beta)$ for every $t \in T$;
- X, α and Y, β are T -similar, written as $X, \alpha \sim_T Y, \beta$, if they induce the same equivalence relation on T : $\text{Val}(t_1, X, \alpha) = \text{Val}(t_2, X, \alpha)$ if and only if $\text{Val}(t_1, Y, \beta) = \text{Val}(t_2, Y, \beta)$ for all $t_1, t_2 \in T$.

In illustration of Kleene Equality remark 2 above, note that if X, Y are coincident/similar for the set T of all \mathcal{Y} -terms, then X, \emptyset and Y, \emptyset are coincident/similar for the set of all extended terms (since the extended terms proper will be undefined under the empty answer function \emptyset).

Proposition 3 (Factorization for Specific Interactions). *Let X, Y be \mathcal{Y} -structures, α, β answer functions for X, Y , respectively, and T a set of extended terms. Then $X, \alpha =_T Y, \beta$ if and only if there is a structure Z and answer function γ for it such that $X, \alpha =_T Z, \gamma \cong Y, \beta$.*

An intrastep interaction variant of the Next Value Theorem is proven in [6, Lemma 8.8]. We shall use a variant adapted to our purpose of relating notions of similarity and indistinguishability:

Theorem 5 (Next Value). *Fix algorithm A . Let X be a state, T a bounded exploration witness and α a context for X . For every ground term t there is a (possibly extended) term $\text{Next}_X^A(t) \in T^t$ such that $\text{Val}(t, \tau_A(X, \alpha)) = \text{Val}(\text{Next}_X^A(t), X)$. Moreover, if β is a context for a state Y and Y, β coincide with X, α over T^t , then $\text{Val}(t, \tau_A(Y, \beta)) = \text{Val}(t, \tau_A(X, \alpha))$.*

As in the non-interactive case, in consequence to Next Value and Factorization we have preservation of coincidence and similarity:

Corollary 6 (Preserving Coincidence and Similarity). *Let X, Y be states and α, β contexts for X, Y , respectively.*

- If X, α and Y, β are coincident (over all extended terms), then $\tau_A(X, \alpha)$ and $\tau_A(Y, \beta)$ are coincident (over all ground terms).
- If X, α and Y, β are similar (over all extended terms), then $\tau_A(X, \alpha)$ and $\tau_A(Y, \beta)$ are similar (over all ground terms).

Reasoning about what an algorithm can do in a state, we will have to take into account all possible behaviors of the environment. Typically we will assume some contract with the environment, there will be assumptions on possible environment behaviors. Thus we define what it means for two structures to be similar for given sets of possible answer functions.

Definition 7 (Similarity under a Contract). Let X, Y be \mathcal{Y} -structures, \mathcal{A}, \mathcal{B} sets of answer functions for X, Y respectively, and T a set of extended terms. We say that X, \mathcal{A} and Y, \mathcal{B} are T -similar, writing $X, \mathcal{A} \sim_T Y, \mathcal{B}$, if

- for every $\alpha \in \mathcal{A}$ there is a $\beta \in \mathcal{B}$ such that $X, \alpha \sim_T Y, \beta$, and
- for every $\beta \in \mathcal{B}$ there is $\alpha \in \mathcal{A}$ such that $X, \alpha \sim_T Y, \beta$.

The idea is again that, by testing terms for equality, an algorithm cannot determine whether it is operating with X, α for some $\alpha \in \mathcal{A}$ or with Y, β for some $\beta \in \mathcal{B}$. If \mathcal{A} resp. \mathcal{B} are seen as representing the degree of freedom that the environment has in fulfillment of its contract, similarity to the notion of bisimulation of transition systems need not be surprising.

Corollary 7 (Factorization under a Contract). *Let X, Y be \mathcal{Y} -structures, \mathcal{A}, \mathcal{B} sets of answer functions for X, Y respectively, and T a set of extended terms. Then $X, \mathcal{A} \sim_T Y, \mathcal{B}$ if and only if*

- for every $\alpha \in \mathcal{A}$ there is $\beta \in \mathcal{B}$, \mathcal{Y} -structure Z and answer function γ over Z such that $X, \alpha =_T Z, \gamma \cong Y, \beta$, and
- for every $\beta \in \mathcal{B}$ there is $\alpha \in \mathcal{A}$, \mathcal{Y} -structure Z and answer function γ over Z such that $Y, \beta =_T Z, \gamma \cong X, \alpha$.

Remark 3 (Contracts). We use a notion of contract heuristically here, we did not define contracts. A proper definition should certainly require that contracts are *abstract*: it should associate a set of answer functions \mathcal{A}_X to any state X in an isomorphism-invariant way. But our results would certainly carry over to such a definition. We are not going to pursue a theory of contracts in this paper.

2.2 Indistinguishability

The notion of indistinguishable states splits here to two notions: states indistinguishable under specific environment behaviors, and states indistinguishable under classes of environment behaviors. We need the former notion in order to formulate the latter.

Definition 8 (Indistinguishability under Specific Interactions). Let X, Y be \mathcal{Y} structures, and α, β answer functions over X, Y respectively, given query templates from E . We say that

- an interactive algorithm A *distinguishes* X, α from Y, β if there is a *ground* \mathcal{Y} -term t such that one of the following holds (but not both):
 - either α is a context for A over X and $Val(t, \tau_A(X, \alpha)) = \mathbf{true}_X$, or if this is not true,
 - β is a context for A over Y and $Val(t, \tau_A(Y, \beta)) = \mathbf{true}_Y$.
- X, α and Y, β are indistinguishable if there is no algorithm distinguishing them.

This definition requires an algorithm, if it is to distinguish X, α from Y, β , to complete its step with at least one of them. Weaker requirements might be argued for, but the intuition that we wish to maintain here is that, in order to distinguish two candidate situations, an algorithm should be able to *determine* that it is running in one of them and not in the other—but in order to determine anything an algorithm must complete its step.

The distinguishing term t is required to be ground. The result of the distinguishing algorithm must be contained in the resulting state, and the value of t in it must not depend on any future interaction. Otherwise, even identical states provided with identical answer functions could be distinguishable.

We also assume that vocabulary of each algorithm contains at least one dynamic function symbol.

Anyway, the choice of this definition is confirmed by the connection to similarity established below. The following corollary is as simple as it was in the previous section:

Corollary 8. *Indistinguishability is an equivalence relation on \mathcal{Y} -structures equipped with E -answer functions.*

Theorem 6. *X, α and Y, β are indistinguishable by interactive algorithms if and only if they are similar.*

Indistinguishability of states for concrete answer functions is thus equivalent to their similarity under the same answer functions. But what we are really interested in is indistinguishability of states for all possible reactions of the environment. The following definition reflects this consideration.

Definition 9 (Indistinguishability under a Contract). Let X and Y be \mathcal{Y} -structures and let \mathcal{A} and \mathcal{B} be sets of answer functions for X and Y , respectively.

- An algorithm A *distinguishes* X, \mathcal{A} from Y, \mathcal{B} if either
 - there is $\alpha \in \mathcal{A}$ such that A distinguishes X, α from Y, β for all $\beta \in \mathcal{B}$, or
 - there is $\beta \in \mathcal{B}$ such that A distinguishes Y, β from X, α for all $\alpha \in \mathcal{A}$.
- X, \mathcal{A} and Y, \mathcal{B} are *indistinguishable* if there is no algorithm distinguishing them.

The intuition here is again that, for an algorithm to distinguish X, \mathcal{A} from Y, \mathcal{B} it must be possible to detect that it is operating in one of them and not in the other. Indistinguishability means here that this is not at all possible, an algorithm can never tell for sure in which of the two worlds it is. It is easy to see that indistinguishability is an equivalence relation on pairs X, \mathcal{A} , where X is an \mathcal{Y} -structure and \mathcal{A} a set of E -answer functions over X .

Corollary 9. *Let X, \mathcal{A} and Y, \mathcal{B} be structures of the same vocabulary, equipped with sets of possible answer functions over the same vocabulary of query-templates. Then they are indistinguishable by interactive ordinary small-step algorithms if and only if they are similar.*

2.3 Accessibility and Reachability

Definition 10 (Accessibility and Reachability under Interaction).

Let x be an element of a state X , Y another state of the same vocabulary with the same carrier, \mathcal{A} a set of answer functions for X and $\alpha \in \mathcal{A}$. We say that

- x is *accessible* for X, α if there is an extended term t denoting it at X, α ;
- x is *accessible* for X, \mathcal{A} if there is $\alpha \in \mathcal{A}$ such that x is accessible for X, α ;
- Y is *reachable* from X, α if there is an algorithm A such that $\tau_A(X, \alpha) = Y$;
- Y is *reachable* from X, \mathcal{A} if there is $\alpha \in \mathcal{A}$ such that Y is reachable from X, α .

Corollary 10 (Accessibility). *If X is a structure and \mathcal{A} a set of answer functions over it, any element of X in the range of an $\alpha \in \mathcal{A}$ is accessible for X, \mathcal{A} .*

Theorem 7. *Let X, Y be structures of a vocabulary \mathcal{Y} with the same base sets and \mathcal{A} be a set of possible answer functions for X . Then Y is reachable from X, \mathcal{A} by ordinary interactive small-step algorithms if and only if*

- $Y - X$ is finite,
- all function symbols occurring in $Y - X$ are dynamic in \mathcal{Y} , and
- there is an $\alpha \in \mathcal{A}$ such that all objects in the common base set occurring in $Y - X$ are also accessible for X, α .

2.4 Algorithms with Import

The idea of modeling creation of new objects, often needed for algorithms, by importing fresh objects from a reserve of naked, amorphous objects devoid of nontrivial properties, has been present in the ASM literature since [10].

An answer function α is *importing* for a state if it has only reserve elements in its range. We specialize notions of accessibility, reachability and indistinguishability under a contract to *importing small-step algorithm*, meaning that answer functions allowed by a contract are importing.

We need the notions and results of the previous sections in particular for algorithms which import new elements, over a background structure [2]. This case is special, since nondeterminism introduced by a choice of reserve element to be imported is inessential up to isomorphism; see [11] for import from a naked set and [2] for import over a background structure.

The reserve of a state was originally defined to be a naked set. In applications, it is usually convenient, and sometimes even necessary, to have some structure like tuples,

sets, lists etc. predefined on *all* elements of a state, including the ones in the reserve. The notion of *background structure* [2] makes precise what sort of structure can exist above a set of atoms without imposing any properties on the atoms themselves, except for their identity.

In this section, we assume that each vocabulary contains a unary predicate *Atomic*. This predicate and the logical constants are called *obligatory* and all other symbols are called *non-obligatory*. The set of atoms of a state X , denoted with $Atoms(X)$, are elements of X for which *Atomic* holds. For reference, we list definitions from [2] below:

Definition 11. A class K of structures over a fixed vocabulary is called a *background class* if the following requirements are satisfied:

BC0 K is closed under isomorphisms.

BC1 For every set U , there is a $X \in K$ with $Atoms(X) = U$.

BC2 For all $X, Y \in K$ and every embedding (of sets) $\zeta : Atoms(X) \rightarrow Atoms(Y)$, there is a unique embedding (of structures) η of X into Y that extends ζ .

BC3 For all $X \in K$ and every $x \in Base(X)$, there is a smallest K -substructure Y of X that contains x .

Suppose that K is a background class. Let S be a subset of a base set of structure $X \in K$. If there is a smallest K -substructure of X containing S , then it is called the *envelope* $E_X(S)$ of S in X and the set of its atoms is called the *support* $Sup_X(S)$ of S in X . In every $X \in K$, every $S \subseteq Base(X)$ has an envelope [2].

A structure X is *explicitly atom-generated* if the smallest substructure of X that includes all atoms is X itself, and a background class BC is explicitly atom-generated if all of its structures are. A background class is *finitary* if the support of every singleton is finite.

Lemma 4. *Every explicitly atom-generated background class is finitary.*

Definition 12 (Backgrounds of Algorithms). We say that a background class K with vocabulary \mathcal{Y}_0 is the *background* of an algorithm A over \mathcal{Y} if

- vocabulary \mathcal{Y}_0 is included in \mathcal{Y} and every symbol in \mathcal{Y}_0 is static in \mathcal{Y} ;
- for every $X \in \mathcal{S}(A)$, the \mathcal{Y}_0 -reduct of X is in K .

The vocabulary \mathcal{Y}_0 is the *background vocabulary* of A , and the vocabulary $\mathcal{Y} - \mathcal{Y}_0$ is the *foreground vocabulary* of A . We say that an element of a state is *exposed*, if it is in a range of a foreground function, or if it occurs in a tuple in the domain of a foreground function. The *active part* of a state is the envelope of the set of its exposed elements and the *reserve* of a state is the set of non-active atoms.

If the algorithm is not fixed, we say that a state X is *over* a background class BC of vocabulary \mathcal{Y}_0 , if \mathcal{Y}_0 -reduct of X is in BC .

The freedom the environment has in choice of reserve elements to import induces *inessential nondeterminism*, resulting in isomorphic states [2]:

Proposition 4. *Every permutation of the reserve of a state can be uniquely extended to an automorphism that is the identity on the active part of the state.*

Intuitively, this means that whatever an algorithm could learn by importing new elements from the reserve does not depend on the particular choice of elements imported. Similarly, one might conjecture that an algorithm cannot learn by importing at all, but this is in general not the case:

Example 1. Up to isomorphism, the non-logical part of a background structure X consists of hereditarily finite sets over its atoms. The only non-obligatory functions are the containment relation \in and a binary relation P : $P(x, y)$ holds in X if $\text{rank}_X(x) = \text{rank}_X(y) + 1$,

where rank_X is defined as:

$$\text{rank}_X(x) = \begin{cases} 0 & \text{if } x \in \text{Atoms}(X) \\ \max\{\text{rank}(y) \mid y \in x\} + 1 & \text{if } x \text{ is a set} \end{cases}.$$

The foreground vocabulary contains only one nullary function symbol f , denoting $\{a\}$ in X and $\{\{a\}\}$ in Y for some atom a (for simplicity, we assume that X and Y have the same reduct over the background vocabulary). Structures X and Y are similar, but for all answer functions α, β evaluating the query \hat{g} to a reserve element, X, α and Y, β are not similar, since $\text{Val}(P(f, g), X, \alpha) = \text{true}$ and $\text{Val}(P(f, g), Y, \beta) = \text{false}$.

By theorem 3 and corollary 9, structures X and Y are indistinguishable by non-interactive small-step algorithms, but distinguishable by small-step algorithm importing from the reserve. Somewhat surprisingly, it follows that import of a reserve element can increase the “knowledge” of an algorithm.

In many common background classes, such as sets, sequences and lists, algorithms *cannot* learn by creation. It is important to have in mind that this property is not guaranteed by the postulates of background classes, and that it must be proved for a concrete background class.

References

1. A. Blass, Y. Gurevich, and S. Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3), 1999.
2. Andreas Blass and Yuri Gurevich. Background, reserve, and Gandy machines. In *Proceedings of CSL '00*, volume 1862 of *LNCS*, 2000.
3. Andreas Blass and Yuri Gurevich. Algorithms: A quest for absolute definitions. *Bulletin of the European Association for Theoretical Computer Science*, (81):195–225, October 2003.
4. Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms I. *ACM Transactions on Computational Logic*, to appear.
5. Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms II. *ACM Transactions on Computational Logic*, to appear.
6. Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms III. *ACM Transactions on Computational Logic*, to appear.
7. Egon Börger and James K. Huggins. Abstract State Machines 1988-1998: Commented ASM bibliography. *Formal Specification Column, EATCS Bulletin*, 64:105–127, 1998.
8. Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58, 1936.
9. Paola Glavan and Dean Rosenzweig. Communicating Evolving Algebras. In *Computer Science Logic*, volume 702 of *LNCS*, pages 182–215. 1993.
10. Yuri Gurevich. Evolving Algebras. A Tutorial Introduction. *Bulletin of the European Association for Theoretical Computer Science*, 43:264–284, 1991.
11. Yuri Gurevich. Evolving Algebras 1993: Lipari Guide. In *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
12. Yuri Gurevich. Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, 2000.
13. Dean Rosenzweig and Davor Runje. Asm model of computational cryptography. *submitted to ASM'07*, 2007. <http://www.fsb.hr/matematika/davor>.
14. Dean Rosenzweig, Davor Runje, and Wolfram Schulte. Model-based testing of cryptographic protocols. In *TGC 2005*, volume 3705 of *LNCS*, pages 33–60. 2005.
15. Robert Stärk and Stanislas Nanchen. A logic for Abstract State Machines. *Universal Journal of Computer Science*, 11(7):981–1006, 2001.
16. Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. In *London Mathematical Society*, volume 42 of 2, pages 230–265, 1936-37.