Evolving MultiAlgebras unify all usual sequential computation models

Serge Grigorieff, Pierre Valarcher

HAL Id: inria-00455373
https://inria.hal.science/inria-00455373

Submitted on 10 Feb 2010

EVOLVING MULTIALGEBRAS
UNIFY ALL USUAL SEQUENTIAL COMPUTATION MODELS

SERGE GRIGORIEFF [1] AND PIERRE VALARCHER [2]

[1] LIAFA, CNRS & Université Paris Diderot - Paris 7, Case 7014, 75205 Paris Cedex 13
  E-mail address: seg@liafa.jussieu.fr
  URL: http://www.liafa.jussieu.fr

[2] LACL, Université de Paris Est, IUT Fontainebleau/Sénart, Route de Hourtault 77300 Fontainebleau
  E-mail address: valarcher@univ-paris12.fr
  URL: http://lacl.univ-paris12.fr/valarcher/

ABSTRACT. It is well-known that Abstract State Machines (ASMs) can simulate "step-by-step" any type of machines (Turing machines, RAMs, etc.). We aim to overcome two facts: 1) simulation is not identification, 2) the ASMs simulating machines of some type do not constitute a natural class among all ASMs. We modify Gurevich's notion of ASM to that of EMA ("Evolving MultiAlgebra") by replacing the program (which is a syntactic object) by a semantic object: a functional which has to be very simply definable over the static part of the ASM. We prove that very natural classes of EMAs correspond via "literal identifications" to slight extensions of the usual machine models and also to grammar models. Though we modify these models, we keep their computation approach: only some contingencies are modified.

Thus, EMAs appear as the mathematical model unifying all kinds of sequential computation paradigms.

CONTENTS

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

1. Introduction

What we prove in this paper. The fact that Abstract State Machines (ASMs) can strict lock-step (i.e. "step-by-step ") simulate any type of machines (Turing machines, stack automata, RAM, etc) and grammars was shown long ago by Gurevich [10, 6]. A systematic study is also done in Börger [2]. A tighter notion of simulation is also valid as shown in Blass, Dershowitz & Gurevich [1].

The questions we consider in this paper are the following:

(Q1) *Can we replace strict lock-step simulation by literal identity (up to a simple change of view)?*

(Q2) *Given a computation model $\mathcal{C}$, is it possible to get a natural characterization of the class of ASMs which are equivalent to machines in $\mathcal{C}$?*

As far as we know, up to now, there is only one isolated answer which is about question (Q2): Gurevich & al. [6] proved that Schönhage Storage Modification Machines correspond exactly (for strict lock-step equivalence) to ASMs with unary functions only.

We bring positive answers to both questions for the diverse usual computation models $\mathcal{C}$ (Turing machines, stack automata, RAMs, Schönhage Machines, Chomsky type 0 grammars, etc.) slightly extended to models $\mathcal{C}^+$ using a tailored version of ASMs which (resurrecting Gurevich's original name for ASMs) we call *Evolving Multialgebras* (EMAs). These answers have the following remarkably simple form:

Theorem 1.1. *There exists a family of EMA static parts $\mathcal{M}$ (fixed semantical feature) and a family of dynamic signatures $\mathcal{S}$ (fixed syntactical feature) such that, letting $\mathcal{E}_{M,S}$ be the family of EMAs with static part in $\mathcal{M}$ and dynamic signature in $\mathcal{S}$,*
- any computation device in $\mathcal{C}^+$ is literally identical to some EMA in $\mathcal{E}_{\mathcal{M},\mathcal{S}}$,
- this "literal identity" correspondence is a bijection from $\mathcal{C}^+$ onto $\mathcal{E}_{\mathcal{M},\mathcal{S}}$.

Of course, *literal identity* is not a formal notion. What we mean is as follows: the diverse components of a computation device in $\mathcal{C}^+$ are in one-one correspondance with the diverse components of the associated EMA, and this correspondance is an identity up to a change of perspective (for instance, a "physical" bi-infinite tape will be considered to be identical to the mathematical set $\mathbb{Z}$ of integers).

Remark 1.2. 1. This theorem is indeed a schema: one theorem per computation model. We have proved it for a variety of usual sequential computation models (cf. [5]).

2. As said above, the diverse instances of Theorem 1.1 are proved for slight extension $\mathcal{C}^+$ of the usual computation models $\mathcal{C}$. In all cases, $\mathcal{C}^+$ can be viewed as $\mathcal{C}$ considered with different time units: for any $k \geq 1$, a device $\mathcal{M}$ in $\mathcal{C}$ is seen as a device $\mathcal{M}^{(k)}$ in $\mathcal{C}^+$ in which one step of $\mathcal{M}^{(k)}$ corresponds to k successive steps of $\mathcal{M}$ (or $< k$ successive steps in case the last of these steps has no successor).

3. Considering another presentation of $\mathcal{C}^+$, one can also view it as $\mathcal{C}$ in which some contingencies have been removed (for instance, the read/write head will be able to scan a window of cells instead of a single cell) but the computational paradigm has been preserved: local computation and a particular topology of data storage for Turing machines, indirect addressing of registers for random access machines, etc. In our opinion, the classes $\mathcal{C}^+$ are the *right ones* to carry the diverse computation paradigms.

4. In fact, contingencies can also be captured by families of EMAs with more technical definitions (cf. [5]): we loose the remarkable simplicity of the above families $\mathcal{E}_{\mathcal{M},\mathcal{S}}$.

5. This theorem schema strengthens Gurevich's claim that ASMs constitute the natural

mathematical modelization of algorithms: *EMAs (which are a variant of ASMs) appear as the computation model which unifies all usual sequential computation paradigms.*

About the proof. No surprise, the proof of Theorem 1.1 for a particular $\mathcal{C}$ involves the particular features of the class $\mathcal{C}$. Thus, the claim (point 5 in Remark 1.2) that Theorem 1.1 is true for extensions $\mathcal{C}^+$ of every usual sequential computation model $\mathcal{C}$ cannot be proved but only be supported by proved instances for a variety of classes $\mathcal{C}$.

As for the common features to all such proofs, they come from an analysis of what precludes positive solutions to questions (Q1) and (Q2). Let us list some of the difficulties which are met. Some are easy to solve, other ones force to adequately tailor the definition of ASMs (as that of EMAs) and those of the usual computation models.

(1) An ASM program mimicking the transition function δ of a Turing machine is a description of δ. Since there are many distinct descriptions of the same δ, there are many ASMs which tightly simulate the same Turing machine. Thus, surprisingly as it is, looking at this component – transition functions –, ASMs are less abstract than Turing machines. Somehow, there is an extra operational feature in ASMs: the operational way to use δ is not part of the formalization of Turing machines.
This is why we modify ASMs to EMAs: *Evolving Multialgebras.* The notion of EMA is that of ASM in which the program (a syntactic object) is replaced by a semantic object: a (very simply definable) functional operating on the function sets over the ASM domain. It is then more natural to break the universe of an ASM into its natural parts: this allows a very useful rudimentary typing of elements and functions.

(3) Again considering Turing machines, an ASM simulates the tape by the set $\mathbb{Z}$ of all integers and the moves of the head by the successor and predecessor operations on $\mathbb{Z}$. Terms in the ASM logical language allow to name the i-th successor and the i-th predecessor. Thus, we cannot avoid the ASM program to move the head more than one cell left or right unless we constrain terms in ASM programs to be of a simple form (somewhat "flat"). Which would put technicalities to any positive answer to question (Q2). This is why we consider slight extensions of the machine models which allow the read/write head to scan a window of cells rather than only one cell and to move in a window. This is a kind of extra capability which is much like allowing several tapes or several heads. Though it does modify the model, it does preserves its core feature: successive local actions.

(4) For machine models having programs like RAMs and SMM (Schönhage Storage Modication Machines), there are two slight modifications. First, allow bounded blocks of parallel and/or successive actions. Second, remove the program and the program counter in favor of a transition function (much in the vein of Turing machines) which, though operating on an infinite set (the contents of the accumulator and of the addressed registers in the case of RAMs) is very simply definable in terms of the original program. Thus, we replace an operational item (the program) by a denotational one (the transition function). Again, though it does modify the model, it does preserves its core feature: indirect addressing (for RAMs), dynamic storage topology (for Schönhage pointer machines).

EMAs versus ASMs. In our opinion, ASMs and EMAs are complementary models. EMAs generalize any type of machine: it is the unification model. As for ASMs, they are closer to programming. Indeed, the functioning of a EMA goes through the iteration of a functional. To program an EMA, we need to add some operational information on how to use this functional and this leads back to a program, hence to an ASM. . . Thus, ASMs are

EMAs plus the instructions for using the functional: ASMs refine EMAs (in the sense of software engineering) and EMAs are a (more) abstract version of ASMs.

2. From ASMs to EMA s: the deterministic case

2.1. How EMAs differ from ASMs

We detail the diverse features which are peculiar to EMAs.

A functional in place of a program. As said in the introduction, the main difference between evolving multialgebras and Gurevich's ASMs is as follows: *the program (i.e. a syntactic object) of an ASM is replaced by a functional (i.e. a semantic object)* which does exactly what the program tells to do. Thus, an operational feature is removed.

Multi-domains and multialgebras. The above modification leads to another very minor one, really kind of "semantic sugar": *the universe of an ASM is broken into its natural constituents and becomes a multi-domain.* The reason for such multialgebras is that they make it possible to type the symbols of the signature as functions (or elements) between the diverse sets of the multi-domain.

Multialgebra operations with values in products of domains. Set theoretically, a map $F : A \to B \times C$ is identified with the pair of its component maps (F_B, F_C) where $F_B : A \to B$ and $F_C : A \to C$. We do view such an F as the pair (F_B, F_C) plus a correlation condition: *one cannot fire one of these two component maps without firing the other one, and both have to be fired on the same argument.*
We allow operations in the multialgebra to take values in products of domains. The above condition leads to a notion of multiterms and a constraint in the definition of formulas associated to the signature of an EMA. It is used in §**??** to deal with Schönhage machines.

Halt/Fail and EMA status. In EMAs, the ASM program is replaced by the functional which does exactly what the program tells to do. There are still the questions:
- is the functional to be applied or not on given arguments?
- if not, does it "halts and accepts" or "halts and rejects" or "get stuck"?
To deal with the three first alternatives, EMAs have a three valued dynamic component: the status. Of course, there is no formal component carrying the information "stuck".

Inputs and ASMs. In most presentations, Gurevich does not give any formal status to inputs (his paper [4] with Dershowitz being an exception). When dealing with question (Q2) it turns out that it is important to give a formal status to inputs. This is the case for EMA characterizations of machines having some read-only tapes (e.g., finite automata).
We consider that *inputs appear in two ways:*
- *as values of some particular static symbols,*
- *as initial values of dynamic symbols.*

2.2. Deterministic Evolving MultiAlgebras

Definition 2.1. Let $n \geq 1$ and $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ be a sequence of n non empty sets (which we call an n-multiset). An n-sort type is a triple (k, α, ℓ) where $k \in \mathbb{N}$, $\ell \in \{1, \ldots, n\}$ and α is a map $\{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$. Its associated $\mathcal{D}$-type $(k, \alpha, \ell)_{\mathcal{D}}$ is the family of all partial functions $D_{\alpha(1)} \times \ldots \times D_{\alpha(k)} \rightarrow D_\ell$. A $\mathcal{D}$-type is functional if $k \geq 1$. In case $k = 0$, the $\mathcal{D}$-type $(0, \emptyset, \ell)_{\mathcal{D}}$ is the family of partial functions $\{\emptyset\} \rightarrow D_\ell$, i.e. the set of "partial elements" of D_ℓ, i.e. D_ℓ augmented with an "undefined element".
Intuition: there are k arguments, α gives their types, and ℓ is the type of the range.
Typed ground terms and their types are defined in the obvious way.

Multialgebras. The notion of multisort algebra is a direct extension to multiset domains of the usual notion of algebra of partial functions on a unique domain.

Definition 2.2 (Multialgebras). Let $n \geq 1$ and $\mathcal{S}$ be an n-sort typed signature containing function symbols $\varphi_1, \ldots, \varphi_p$. An $\mathcal{S}$-multialgebra $\mathcal{A}$ is an n-multiset $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ endowed with partial functions $F_1, \ldots, F_p$ which interpret the symbols φ_i's (Care: arity 0 symbols with type D_i are interpreted by elements of D_i but can also be undefined).
If defined, the value, relative to $\mathcal{A}$, of a ground $\mathcal{S}$-term t is denoted by $[\![t]\!]_{\mathcal{A}}$ (it is an element of some D_i).

Semialgebraic functionals. Semialgebraic functionals are those which can be described by ASM programs. They modify the interpretations in the multialgebra of constant and functions symbols. For function symbols, this modification affects the values of only finitely many points in the domain. These points and the associated new values of the argument are given by ground $\mathcal{S}$-terms. As in ASMs programs, there is a disjunction of cases for the choice of the affected points and their associated new values.
    First, a convenient notion.

Definition 2.3 (The $\oplus$ operation). Let F, G be partial functions $X_1 \times \ldots \times X_k \rightarrow Y$ and $Z \subseteq X_1 \times \ldots \times X_k$. We define the partial function $F \oplus_Z G$ as follows:
$$\text{Domain}(F \oplus_Z G) = (\text{Domain}(F) \setminus Z) \cup (\text{Domain}(G) \cap Z)$$
$$(F \oplus_Z G)(\vec{x}) = \begin{cases} F(\vec{x}) & \text{if} \quad \vec{x} \notin Z \\ G(\vec{x}) & \text{if} \quad \vec{x} \in Z \end{cases}$$
In case $p = 0$, F, G are "partial elements" of Y and $Z \subseteq \{\emptyset\}$ and $F \oplus_Z G = F$ if $Z = \emptyset$ and $F \oplus_Z G = G$ if $Z = \{\emptyset\}$.

Definition 2.4 (Semialgebraic functionals). Let
• $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ be an n-multiset,
• $\mathcal{S}$ be an n-sort typed signature containing function symbols $\varphi_1, \ldots, \varphi_p$,
• $\mathcal{A}$ be a multialgebra with signature $\mathcal{S} \setminus \{\varphi_1, \ldots, \varphi_p\}$ on $\mathcal{D}$,
• $\mathcal{F}_1, \ldots, \mathcal{F}_p$ be the $\mathcal{D}$-types associated to $\varphi_1, \ldots, \varphi_p$,
• $m \in \{1, \ldots, p\}$ and (k, α, ℓ) be the n-sort type of φ_m.
• $\mathcal{T}_i$ be the family of ground $\mathcal{S}$-terms of type D_i,
For any p-tuple of functions $\vec{F} = (F_1, \ldots, F_p) \in \mathcal{F}_1 \times \ldots \times \mathcal{F}_p$, let us denote by $\mathcal{A}(\vec{F})$ the multialgebra $\mathcal{A}$ expanded to the signature $\mathcal{S}$ in which the φ_i's are interpreted by the F_i's.
A partial functional $\Phi : \prod_{j=1,\ldots,p} \mathcal{F}_j \longrightarrow \mathcal{F}_m$ is $(\mathcal{S}, \mathcal{A})$-semialgebraic if there exists a map $\beta : \text{Bool}^q \rightarrow \mathfrak{P}_{fin}(\mathcal{T}_{\alpha(1)} \times \ldots \times \mathcal{T}_{\alpha(k)} \times \mathcal{T}_\ell)$ (where $\mathfrak{P}_{fin}(X)$ is the family of finite subsets of

X) and ground $\mathcal{S}$-terms $t_1, \ldots, t_q,\, t_1'\ldots, t_q'$ such that, for any $\vec{F} \in \mathcal{G}_1 \times \ldots \times \mathcal{G}_p$,

$\Phi(\vec{F})$ is defined if and only if
$$\begin{cases} (a) & \text{all } [\![t_i]\!]_{\mathcal{A}(\vec{F})}\text{'s, } [\![t_i']\!]_{\mathcal{A}(\vec{F})}\text{'s are defined} \\ (b) & \forall (u_1, \ldots, u_k, v) \in \beta(\ldots, [\![t_i]\!]_{\mathcal{A}(\vec{F})} = [\![t_i']\!]_{\mathcal{A}(\vec{F})}, \ldots) \text{ all } [\![u_j]\!]_{\mathcal{A}(\vec{F})}\text{'s are defined} \\ (c) & \forall (\vec{u}, v), (\vec{w}, z) \in \beta(\ldots, [\![t_i]\!]_{\mathcal{A}(\vec{F})} = [\![t_i']\!]_{\mathcal{A}(\vec{F})}, \ldots) \quad [\![u_j]\!]_{\mathcal{A}(\vec{F})} \neq [\![w_j]\!]_{\mathcal{A}(\vec{F})} \text{ for some } j \end{cases}$$

$\Phi(\vec{F}) = F_m \oplus_Z G$ where
$Z = \{([\![u_1]\!]_{\mathcal{A}(\vec{F})}, \ldots, [\![u_k]\!]_{\mathcal{A}(\vec{F})}) \mid \exists v\ (\vec{u}, v) \in \beta(\ldots, [\![t_i]\!]_{\mathcal{A}(\vec{F})} = [\![t_i']\!]_{\mathcal{A}(\vec{F})}, \ldots)\}$
$G = \{([\![u_1]\!]_{\mathcal{A}(\vec{F})}, \ldots, [\![u_k]\!]_{\mathcal{A}(\vec{F})}, [\![v]\!]_{\mathcal{A}(\vec{F})}) \mid (\vec{u}, v) \in \beta(\ldots, [\![t_i]\!]_{\mathcal{A}(\vec{F})} = [\![t_i']\!]_{\mathcal{A}(\vec{F})}, \ldots)\}$

The tuple $(\beta, t_1, \ldots, t_q, t_1' \ldots, t_q')$ is called a presentation of Φ.

For $I \subseteq \{1, \ldots, p\}$, a functional $\Psi : \prod_{j=1,\ldots,p} \mathcal{F}_j \longrightarrow \prod_{m \in I} \mathcal{F}_m$ is $(\mathcal{S}, \mathcal{A})$-semialgebraic if so are all its components.

Remark 2.5. Condition (a) in Definition 2.4 insures that all equality tests $t_i = t_i'$ can be achieved. Conditions (b) and (c) insure that, in equality $\Phi(\vec{F}) = F_m \oplus_Z G$, the finite set Z can be computed and G is a functional graph.
We do not require the $[\![v]\!]_{\mathcal{A}(\vec{F})}$'s to be defined (i.e. $\texttt{Domain}(G) = Z$): though this is incompatible with a call by value strategy, it makes sense with a call by name strategy.

Definition 2.6 (**Deterministic EMAs**). A deterministic evolving multialgebra (EMA) is a tuple $\mathcal{A} = (n;\ \mathcal{S}_{\text{sta}}, \mathcal{S}_{\text{input}}^{\text{sta}}, \mathcal{S}_{\text{input}}^{\text{dyn}}, \mathcal{S}_{\text{dyn}};\ \mathcal{D};\ \mathcal{M}_{\text{sta}}, \mathcal{M}_{\text{ini}};\ \Phi)$ consisting of the following items.

- An n-multiset $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ such that $D_n = \{\text{go}, \text{acc}, \text{rej}\}$.
  Intuition. Sets $D_1, \ldots, D_{n-1}$ are the $n-1$ different sorts of objects and $D_n = \{go, acc, rej\}$ is the set of possible statuses of the (evolving) multialgebra during the run: "go on", "halt and accept", "halt and reject".
- Four disjoint n-sort typed finite signatures $\mathcal{S}_{\text{sta}}, \mathcal{S}_{\text{input}}^{\text{sta}}, \mathcal{S}_{\text{input}}^{\text{dyn}}, \mathcal{S}_{\text{dyn}}$ and two structures $\mathcal{M}_{\text{sta}}, \mathcal{M}_{\text{ini}}$ with respective signatures $\mathcal{S}_{\text{sta}}, \mathcal{S}_{\text{dyn}}$. There is only one symbol $\mathfrak{s}$ which involves the sort n : it is a constant of type D_n in $\mathcal{S}_{\text{input}}^{\text{dyn}}$.
  Intuition. $\mathcal{M}_{sta}$ is the static framework on $\mathcal{D}$ which remains fixed during any run. $\mathcal{S}_{input}^{sta}$ is the signature for the static part of the input: its interpretation remains fixed (hence accessible) during a run. $\mathcal{S}_{input}^{dyn}$ is the signature for the dynamic part of the input: its interpretation can be modified (hence become inaccessible) during a run. $\mathcal{M}_{ini}$ initializes the part of the dynamic environment which is not initialized by the input. The interpretation of $\mathfrak{s}$ represents the status of the multialgebra.
- Let $\mathcal{S} = \mathcal{S}_{\text{sta}} \cup \mathcal{S}_{\text{input}}^{\text{sta}} \cup \mathcal{S}_{\text{input}}^{\text{dyn}} \cup \mathcal{S}_{\text{dyn}}$. Φ is a $(\mathcal{S}, \mathcal{M}_{\text{sta}})$-semialgebraic partial functional

$$\Phi : \left(\prod_{\varphi \in \mathcal{S}_{\text{input}}^{\text{sta}}} \mathcal{F}_\varphi \right) \times \left(\{\text{go}\} \times \prod_{\varphi \in (\mathcal{S}_{\text{dyn}} \cup \mathcal{S}_{\text{input}}^{\text{dyn}}) \setminus \{\mathfrak{s}\}} \mathcal{F}_\varphi \right) \longrightarrow \prod_{\varphi \in \mathcal{S}_{\text{dyn}} \cup \mathcal{S}_{\text{input}}^{\text{dyn}}} \mathcal{F}_\varphi$$

  where $\mathcal{F}_\varphi$ denotes the semantic type of the function symbol φ. In particular, Φ rules the evolution of the status. The sole status which can be an argument of Φ is "go": a multialgebra with status "acc" or "rej" is halted and does not evolve any more. However, in the image of Φ the status can take any value.

A state of $\mathcal{A}$ is any multialgebra on $\mathcal{D}$ with signature $\mathcal{S}$ which expands $\mathcal{M}_{\mathrm{sta}}$.

Definition 2.7 (Runs of deterministic EMAs). We keep the notations of Definition 2.6. A run of $\mathcal{A}$ is a sequence $(\mathcal{M}_t)_{t \in I}$ of states of $\mathcal{A}$ such that
- I is a finite or infinite non empty initial segment of $\mathbb{N}$,
- $[\![\theta]\!]_{\mathcal{M}_0} = [\![\theta]\!]_{\mathcal{M}_{\mathrm{ini}}}$ for all $\theta \in \mathcal{S}_{\mathrm{dyn}}$,
- If $t \in I$ then $[\![\theta]\!]_{\mathcal{M}_t} = [\![\theta]\!]_{\mathcal{M}_0}$ for all $\theta \in \mathcal{S}_{\mathrm{input}}^{\mathrm{sta}}$,
- If $t \in I$ then $t+1$ is in I if and only if $[\![\mathfrak{s}]\!]_{\mathcal{M}_t} = $ go and Φ is defined on $([\![\varphi]\!]_{\mathcal{M}})_{\varphi \in \mathcal{S} \setminus \mathcal{S}_{sta}}$,
- If $t+1 \in I$ then $([\![\theta]\!]_{\mathcal{M}_{t+1}})_{\theta \in \mathcal{S}_{\mathrm{dyn}} \cup \mathcal{S}_{\mathrm{input}}^{\mathrm{dyn}}} = \Phi(([\![\varphi]\!]_{\mathcal{M}_t})_{\varphi \in \mathcal{S} \setminus \mathcal{S}_{sta}})$.

In particular, if $[\![\mathfrak{s}]\!]_{\mathcal{M}_0} \neq$ go then $I = \{0\}$. Also, if $t+1 \in I$ then $[\![\mathfrak{s}]\!]_{\mathcal{M}_t} = $ go.

3. Turing machines

In order to *identify* Turing machines with a simple class of EMAs, we introduce a slight variant of Turing machines, which we call "window Turing machines": 1) the head is allowed to scan a small window instead of a single cell, and to move inside a window in a single step, 2) halting (be it accepting or rejecting) is not related to the current state but to the current local configuration: the state plus the contents of the scanned window.

Definition 3.1. A deterministic k-window n-tape (bi-infinite tapes) Turing machine is a tuple $(n, k, \Sigma = \{\sigma_0, \ldots, \sigma_{s-1}\}, Q = \{q_0, \ldots, q_{r-1}\}, F^+, F^-, \delta, \omega_i, \mu_i)_{i=1,\ldots,n}$ where, for $i = 1, \ldots, n$,

- Σ and Q are finite sets (the alphabet and the set of states),
- $F^+, F^- \subseteq Q \times \Sigma^{n(2k+1)}$ (accepting/rejecting final local configurations),
- $\delta : Q \times \Sigma^{n(2k+1)} \to Q$ (state transition),
- $\tau_i : Q \times \Sigma^{n(2k+1)} \to \Sigma^{n(2k+1)}$ (read/write on tape i),
- $\mu_i : Q \times \Sigma^{n(2k+1)} \to \{-k, \ldots, -1, 0, 1, \ldots, k\}$ (move on tape i).

On each tape, the head scans the cell on which it is positioned and the k cells to the left and the k cells to the right, a total of $2k+1$ cells. The argument of type $\Sigma^{n(2k+1)}$ in δ, ω_i, μ_i is the contents of the $n(2k+1)$ cells scanned on the n tapes. The effect of a transition is to change the state according to δ, to modify the contents of the scanned cells of tape i according to ω_i and to move its head according to μ_i.

The notions of run, halt, acceptance and rejection are defined as usual.

Remark 3.2. Usual deterministic n-tape Turing machines are the 1-window ones.

Definition 3.3 (The class of EMAs for Turing machines). We denote by $\mathcal{C}_{wT}^{(n)}$ the class of EMAs $\mathcal{A} = (n+3; \mathcal{S}_{\mathrm{sta}}, \mathcal{S}_{\mathrm{input}}^{\mathrm{sta}}, \mathcal{S}_{\mathrm{input}}^{\mathrm{dyn}}, \mathcal{S}_{\mathrm{dyn}}; \mathcal{D}; \mathcal{M}_{\mathrm{sta}}, \mathcal{M}_{\mathrm{ini}}; \Phi)$ which satisfy the following conditions for some $r, s \in \mathbb{N}$ (for clarity, we abusively denote by the same letter static constant symbols and the elements which interpret them in the structure $\mathcal{D}$).

(1) The multidomain of $\mathcal{A}$ is $\mathcal{D} = (\mathbb{Z}^{(1)}, \ldots, \mathbb{Z}^{(n)}, Q, \Sigma, \mathfrak{S})$ where the $\mathbb{Z}^{(i)}$'s are fixed pairwise disjoint copies of $\mathbb{Z}$ (for instance, $\mathbb{Z}^{(i)} = \mathbb{Z} \times \{i\}$), Q, Σ are finite sets with r, s elements respectively, and $\mathfrak{S} = \{\mathrm{go}, \mathrm{acc}, \mathrm{rej}\}$.

(2) The static framework signature $\mathcal{S}_{\mathrm{sta}}$ contains r constants $q_0, \ldots, q_{r-1}$ of type Q, s constants $\sigma_0, \ldots, \sigma_{s-1}$ of type Σ and three constants $\mathrm{go}, \mathrm{acc}, \mathrm{rej}$ of type $\mathfrak{S}$ which are interpreted in the obvious way in $\mathcal{M}_{\mathrm{sta}}$. It also contains, for each $i = 1, \ldots, n$, two unary functions

symbols $Succ^{(i)}, Pred^{(i)}$ of type $\mathbb{Z}^{(i)} \to \mathbb{Z}^{(i)}$ which are interpreted in $\mathcal{M}_{\mathrm{sta}}$ as the successor and predecessor functions in $\mathbb{Z}^{(i)}$.

(3) The signature $\mathcal{S}^{\mathrm{sta}}_{\mathrm{input}}$ is empty.

(4) The signature $\mathcal{S}_{\mathrm{dyn}}$ (for the dynamic environment non initialized by the input) contains, for each $i = 1, \dots, n$, one constant $\mathrm{pos}^{(i)}$ of type $\mathbb{Z}^{(i)}$ one constant q of type Q, and one constant $\mathfrak{s}$ of type $\mathfrak{S}$, which are respectively interpreted in $\mathcal{M}_{\mathrm{ini}}$ as 0, q_0 and go.

(5) The signature $\mathcal{S}^{\mathrm{dyn}}_{\mathrm{input}}$ (for the dynamic environment initialized by the input) contains, for each $i = 1, \dots, n$, one unary function $c^{(i)}$ of type $\mathbb{Z}^{(i)} \to \Sigma$.

Thus, the EMAs in $\mathcal{C}^{(n)}_{wT}$ are defined as those having particular signature, multidomain, static framework and initialization of some dynamic symbols with no condition on the functional Φ (other than its semialgebraicity).

Theorem 3.4 (**EMA representation theorem for Turing machines**).
Any deterministic n-tape window Turing machine is literally identical *to some EMA in the class $\mathcal{C}^{(n)}_{wT}$. Conversely, any EMA in $\mathcal{C}^{(n)}_{wT}$ is* literally identical *to some deterministic n-tape window Turing machine.*

Proof. The argument is based on the following literal identifications between the components of a Turing machine (TM) and the interpretations of symbols of the EMA signature:
   (1) (TM) i-th tape and the way the read/write head moves on it.
       (EMA) the copy $\mathbb{Z}^{(i)}$ of $\mathbb{Z}$ structured as $\langle \mathbb{Z}^{(i)}, Succ^{(i)}, Pred^{(i)} \rangle$.
   (2) (TM) diverse states and letters.
       (EMA) interpretations of the static symbols $q_0, \dots, q_{r-1}$ and $\sigma_0, \dots, \sigma_{s-1}$.
   (3) (TM) current state, positions of the n heads and contents of the n tapes.
       (EMA) current interpretations of the dynamic symbols q, $\mathrm{pos}^{(i)}$, $c^{(i)}$.
   (4) (TM) non final or final accepting/rejecting character of the current state.
       (EMA) current interpretation of the dynamic symbol $\mathfrak{s}$.
   (5) (TM) transition function.
       (EMA) semialgebraic functional.
   (6) (TM) initial configuration.
       (EMA) interpretations of the c_i's in the initial multialgebra and of $\mathcal{S}^{\mathrm{dyn}}_{\mathrm{dyn}}$ in $\mathcal{M}_{\mathrm{ini}}$.
The non trivial identifications are those of points 4 and 5.
Keeping the notations of Definition 2.4, let $(\beta_\varphi, t_{1,\varphi}, \dots, t_{q_\varphi,\varphi}, t'_{1,\varphi} \dots, t'_{q_\varphi,\varphi})_{\varphi \in \mathcal{S}^{\mathrm{int}}_{\mathrm{dyn}}}$ be a presentation of the semialgebraic functional Φ of an EMA:

$$\beta_\varphi : \mathtt{Bool}^{q_\varphi} \to \mathfrak{P}_{fin}(\mathcal{T}_{\alpha_\varphi(1)} \times \dots \times \mathcal{T}_{\alpha_\varphi(k_\varphi)} \times \mathcal{T}_{\ell_\varphi})$$

Observe that terms of type $\mathbb{Z}^{(j)}$ are of the form $\xi_1(\xi_2(\dots))(\mathrm{pos}^{(j)})$ where the ξ_k's are $Succ^{(j)}$ or $Pred^{(j)}$. Let k be the maximum value of the $|\xi_1(\xi_2(\dots))(0)|$ for all terms of type some $\mathbb{Z}^{(j)}$ which is among the $t_{i,\varphi}, t'_{i,\varphi}$ or among the finite sets given by the β_φ's.
   First, let us look at the equalities $t_{i,\varphi} = t'_{i,\varphi}$ which govern the domain of Φ.
• If $t_{i,\varphi}, t'_{i,\varphi}$ have type $\mathbb{Z}^{(j)}$ then, as said above, they are of the form $\xi_1(\xi_2(\dots))(\mathrm{pos}^{(j)})$. Hence any equality $t_{i,\varphi} = t'_{i,\varphi}$ is trivially true or false independently of the current value of $\mathrm{pos}^{(j)}$.
If $t_{i,\varphi}, t'_{i,\varphi}$ have type $\mathfrak{S}$ then they are of the form $\mathfrak{s}$ or go, acc, rej. Since Φ and β are restricted

to values where $\mathfrak{s} = \mathrm{go}$, all possible equalities are trivial.

Thus, we can suppose that there is no term with type $\mathbb{Z}^{(j)}$ or $\mathfrak{S}$ among the $t_{i,\varphi}, t'_{i,\varphi}$'s.

• If $t_{i,\varphi}, t'_{i,\varphi}$ have type Q then they are of the form q or q_j ($j = 0, \ldots, r-1$). Since any equality $q_j = q_k$ is trivially true or false, we can suppose that there is at most one equality between terms of type Q and that it is of the form $q = q_j$.

• If $t_{i,\varphi}, t'_{i,\varphi}$ have type Σ then they are of the form $c^{(j)}(\xi_1(\xi_2(\ldots))(\mathrm{pos}^{(j)}))$ where the ξ_k's are $Succ^{(j)}$ or $Pred^{(j)}$. The equalities between terms of type Σ are all comparisons of letters among the values of $c^{(1)}(-k), \ldots, c^{(1)}(k), \ldots, c^{(n)}(-k), \ldots, c^{(n)}(k)$ where k is defined above.

This shows that the values of Φ depend solely on the value of q and those of the $c^{(j)}(\mathrm{pos}^{(j)} + i)$'s for $j = 1, \ldots, n$ and $i = -k, \ldots, k$). This is exactly to say that what matters is the current state and the current letters in the n windows of diameter $2k+1$ centered at the positions of the n heads. Otherwise said, the tuple of arguments of the functional Φ is literally identical to the current values of the state plus the contents of the windows, that is a tuple in $Q \times \Sigma^{n(2k+1)}$.

Let us look at the image of Φ which is given through finite families of tuples of terms given by the β_φ's. Since the only terms of type Q are q and the q_i's. Thus, Φ can leave the dynamic symbol q unchanged or modify it to any value. The same is valid for the dynamic symbol $\mathfrak{s}$ (using what is said above about the domain of Φ, this proves the non easy direction of point 4).

Terms of type Σ name the contents of some $c^{(j)}$ at positions which are at distance $\leq k$ of the position of the j-th head. Thus Φ can modify the values of the $c^{(j)}$ in the windows around the positions of the heads.

Terms of type $\mathbb{Z}^{(j)}$ name an integer at distance k of the position of the j-th head. Thus Φ can move any head left or right of at most k cells. This proves the non easy direction of point 5. Thus, an EMA in $\mathcal{C}_T^{(n)}$ is literally identical to some window Turing machine. The converse is proved in a similar (much easier) way. ∎

Remark 3.5. A slight variation in the EMA model can have strong effect. For instance, suppose we add a constant 0 to the static signature and interpret it as 0 in the structure $\mathcal{M}_{\mathrm{sta}}$. Then we get window Turing machines in which the head can jump to cell 0.

4. Random access machines

In order to *identify* RAMs with a simple class of EMAs, we introduce a slight variant of RAMs, which we call "transition RAM" (TRAM): 1) a bounded number of registers can be modified in one step, 2) it can test for equality to 0 and equality between combinations (via the fixed set of operations on $\mathbb{N}$) of the contents of the addressed registers, 3) the program is replaced by a transition function. Though this function operates on an infinite domain, it is finitarily defined via ground terms.

Definition 4.1 (*n*-transition RAMs). Let $f_1, \ldots, f_p$ operations on non negative integers, A n-transition RAM (n-TRAM) with operations $f_1, \ldots, f_p$ is a tuple

$$(n, k, Q = \{q_0, \ldots, q_{r-1}\}, F^+, F^-, \delta, \rho_i, \tau_{i,j})_{i=1,\ldots,n,\ j=1,\ldots,k}$$

where

- n is the number of distinguished registers,
- Σ and Q are finite sets (the alphabet and the set of states),

- $F^+, F^- \subseteq Q \times \mathtt{Bool}^p$ (accepting/rejecting final local configurations),
- $\delta : Q \times \mathtt{Bool}^p \to Q$ (state transition),
- $\rho_i : Q \times \mathtt{Bool}^p \to T$ (modification of register i) for $i = 1, \ldots, n$, where T is a finite family of terms built with the operations $f_1, \ldots, f_p$ and $n(1+k)$ constants (representing the contents of the addressed registers),
- $\tau_{i,j} : Q \times \mathtt{Bool}^p \to T$ (modification of the register addressed through an iteration of j successive addressing, starting with register i), for $i = 1, \ldots, n$, $j = 1, \ldots, k$.

At any time the n-TRAM accesses registers $1, \ldots, n$ and the registers addressed addressed through at most k iterated addressing by these registers. The $p = n(1+k)(1 + \frac{n(1+k)-1}{2})$ Boolean arguments in the δ, ρ_i, τ_i's test equalities or equalities to 0 of the contents of the $n(1+k)$ adressed registers. Map δ tells how the state is modified. Maps $\rho_i, \tau_{i,j}$'s tell how the contents of the accessed registers are modified.

The notions of run, halt, acceptance and rejection are defined in the usual way.

Definition 4.2 (The class of EMAs for TRAMS). Let $f_1, \ldots, f_p$ operations on non negative integers. We denote by $\mathcal{C}^{(n)}_{\mathrm{TRAM}}$ the class of EMAs $\mathcal{A}$ which satisfy the following conditions.

(1) $\mathcal{A}$ has 4 sorts and its multidomain is $\mathcal{D} = (\mathbb{N}, \mathbb{N}^{\mathrm{addr}}, Q, \mathfrak{S})$ where $\mathbb{N}^{\mathrm{addr}}$ is a copy of $\mathbb{N}$, Q is a finite set with r elements, and $\mathfrak{S} = \{\mathrm{go}, \mathrm{acc}, \mathrm{rej}\}$.

(2) The signature $\mathcal{S}_{\mathrm{sta}}$ (for the static framework) contains $n + r + 3$ constants: $1, \ldots, n$ of type $\mathbb{N}$, $q_0, \ldots, q_{r-1}$ of type Q, "go", "acc", "rej" of type $\mathfrak{S}$, and $n + 1$ unary function symbols *cast* of type $\mathbb{N} \to N^{\mathrm{addr}}$, and, for each $i = 1, \ldots, n$, f_i of type $\mathbb{N}^{k_i} \to \mathbb{N}$. Their interpretations in $\mathcal{M}_{\mathrm{sta}}$ are as follows: i) f_i is interpreted as the given operation on $\mathbb{N}$, ii) the cast function is interpreted as the identity from $\mathbb{N}$ to its copy N^{addr}, iii) $1, \ldots, n$, the q_i's and "go", "acc", "rej" are interpreted in the obvious way.

(3) The signature $\mathcal{S}^{\mathrm{sta}}_{\mathrm{input}}$ is empty.

(4) The signature $\mathcal{S}_{\mathrm{dyn}}$ contains two constants $q, \mathfrak{s}$ of types Q and $\mathfrak{S}$. Their interpretations in $\mathcal{M}_{\mathrm{ini}}$ are q_0 and "go".

(5) The signature $\mathcal{S}^{\mathrm{dyn}}_{\mathrm{input}}$ contains one unary function c of type $\mathbb{N}^{\mathrm{addr}} \to \mathbb{N}$.

Thus, the EMAs in $\mathcal{C}^{(n)}_{\mathrm{TRAM}}$ are defined as those having particular signature, multidomain, static framework and initialization of some dynamic symbols with no condition on the functional Φ (other than its semialgebraicity).

Theorem 4.3 (EMA representation theorem for TRAMs).
Any n-TRAM is literally identical to some EMA in the class $\mathcal{C}^{(n)}_{TRAM}$. Conversely, any EMA in $\mathcal{C}^{(n)}_{TRAM}$ is literally identical to some n-TRAM.

Proof. Analogous to the proof of Theorem 3.4.   ∎

5. Other models

Similar results can be proved with finite atomata, stack automata Schönhage machines.

Let us mention an interesting feature occurring in the EMA modelization of Schönhage Storage Modification Machines (SMM) which illustrates what has been said in §2.1 about

operations with values in products of domains. The tape of an SMM is a dynamic graph which may grow or loose nodes. To manage the current set of nodes of this graph-tape, it is convenient to introduce the following items:

• Among the sets of the multi-domain $\mathcal{D}$, there is an infinite set X (where all nodes are taken) and the set $\mathfrak{P}_{fin}(X)$ of finite subsets of X. There is no structure on X nor on $\mathfrak{P}_{fin}(X)$.

• In the signature $\mathcal{S}_{\mathrm{dyn}}$, there is a constant symbol U of type $\mathfrak{P}_{fin}(X)$ (it tells which nodes are in the current graph-tape).

• In the signature $\mathcal{S}_{\mathrm{sta}}$, there is a function symbol *new* with type $\mathfrak{P}_{fin}(X) \to X \times \mathfrak{P}_{fin}(X)$. It is interpreted as a choice function $A \mapsto (a, A \cup \{a\})$ which picks in X a point outside A, i.e. such that $a \notin A$.

To add a new node to the graph tape, we apply *new* to U. The constraint that both components of *new* have to be fired simultaneously and on the same argument insures that when a new node is picked, it is automatically added to (the interpretation) of U with no condition on the functional Φ.

6. Uniformly bounded non determinism

Uniformly bounded non determinism allows at each step at most k choices where k is some fixed constant independent of the step. EMAs with 'such non determinism are defined as are deterministic EMAs with the following modification: *replace the semialgebraic functional Φ by finitely many such functionals.* All litteral identity results mentioned in the previous sections extend easily to the non deterministic cases.

7. External non determinism

We now deal with a more powerful kind of non determinism: that given by external choices which may be done during the run. This is the action of Gurevich's "Choose" instruction. To deal with such an "external non determinism", we enrich EMAs with a fifth signature: the "external dynamic" signature $\mathcal{S}_{\mathrm{ext}}$. We illustrate this notion with the example of Chomsky type 0 grammars.

Definition 7.1. A grammar is a finite set of rules $(u_i, v_i)_{i=1,\ldots,n}$ where the u_i, v_i's are words in an alphabet Σ. The associated relation $R \subseteq \Sigma^\star \times \Sigma^\star$ is defined as follows: a pair (U, V) is in R if and only if there exists a finite sequence $U = U_0, \ldots, U_k = V$ such that, for all $j < k$ there exists words P, S and some $i = 1, \ldots, n$ such that $U_j = Pu_iS$ and $U_{j+1} = Pv_iS$.

Definition 7.2. We denote by $\mathcal{C}_{\mathrm{gra}}$ the class of non deterministic EMAs

$$\mathcal{A} = (3; \ \mathcal{S}_{\mathrm{sta}}, \mathcal{S}_{\mathrm{input}}^{\mathrm{sta}}, \mathcal{S}_{\mathrm{input}}^{\mathrm{dyn}}, \mathcal{S}_{\mathrm{dyn}}, \mathcal{S}_{\mathrm{ext}}; \ \mathcal{D}; \ \mathcal{M}_{\mathrm{sta}}, \mathcal{M}_{\mathrm{ini}}; \ \Phi)$$

which satisfy the following conditions.

(1) $\mathcal{A}$ has 3 sorts and its multidomain is $\mathcal{D} = (\mathbb{N}, \Sigma^*, \mathfrak{S})$ where Σ is a finite set.

(2) The signature $\mathcal{S}_{\mathrm{sta}}$ (for the static framework) contains finitely many binary function symbols $subst_i$, $i = 1, \ldots, n$ of type $\mathbb{N} \times \Sigma^* \to \Sigma^*$. There is some family $(u_i, v_i)_{i=1,\ldots,n}$ of pairs of words such that the interpretation in $\mathcal{M}_{\mathrm{sta}}$ (the static framework) of $subst_i$ is the function which acts on a pair (p, U) as follows: if U contains the factor u_i in position p then it is replaced by v_i, else U is not modified.

(3) The signatures $\mathcal{S}_{\mathrm{input}}^{\mathrm{sta}}$ and $\mathcal{S}_{\mathrm{dyn}}$ are empty.

(4) The signature $\mathcal{S}_{\text{input}}^{\text{dyn}}$ contains one constant w of type Σ^*.

(5) The signature $\mathcal{S}_{\text{ext}}$ (the external dynamic environment) contains one constant *Choose* of type $\mathbb{N}$. Its interpretation during the run is given as an external action: its value changes at each step.

Thus, the EMAs in $\mathcal{C}_{\text{gra}}^{(n)}$ are defined as those having particular signature, multidomain, static framework and initialization of some dynamic symbols with no condition on the functional Φ (other than its semialgebraicity).

Using the fact that iteration of substitutions is also a substitution, one can prove :

Theorem 7.3. *Any grammar is* literally identical *to some EMA in the class $\mathcal{C}_{gra}$. Conversely, any EMA in $\mathcal{C}_{gra}$ is* literally identical *to some grammar.*

References

[1] Andreas Blass, Nachum Dershowitz and Yuri Gurevich. Exact exploration. *Microsoft TechReport MSR-TR-2009-99*, 2009.

[2] Egon Börger. Unifying View of Models of Computation and System Design Frameworks. *Annals of Pure and Applied Logic*, 133: 149-171, 2005.

[3] Giuseppe Del Castillo and Yuri Gurevich and Karl Stroetmann. Typed Abstract State Machines. *Unfinished manuscript*, 25 pages, 1998.

[4] Nachum Dershowitz and Yuri Gurevich. A natural axiomatization of computability and proof of Church's Thesis. *Bulletin. of Symbolic Logic*, 14(3):299–350, 2008.

[5] Serge Grigorieff and Pierre Valarcher. Evolving MultiAlgebras unify all usual sequential computation models. `http://lacl.univ-paris12.fr/valarcher/`.

[6] S. Dexter and P. Boyle and Y. Gurevich. Gurevich Abstract State Machines and Schönhage Storage Modification Machines. *JUCS*, 3(4): 279–303, 1997.

[7] Yuri Gurevich. Reconsidering Turing's Thesis: towards more realistic semantics of programs. *Technical Report CRL-TR-38-84, EEC Dept, Univ. Michigan*, 1984.

[8] Yuri Gurevich. A new Thesis. *Abstracts, American Math. Soc.*, 1985.

[9] Yuri Gurevich. *Logic and the Challenge of Computer Science*. Current Trends in Theoretical Computer Science, ed. Egon Börger, Computer Sc. Press. 1–57, 1988.

[10] Yuri Gurevich. Evolving Algebras: An Introductory Tutorial. *Bul. EATCS*, 43: 264–284, 1991. Reprinted in *Current Trends in Theoretical Computer Science, 1993*, 266–29, World Scientific, 1993.

[11] Yuri Gurevich. *May 1997 Draft of the ASM Guide*. Tech Report CSE-TR-336-97, EECS Dept, University of Michigan, 1997.

[12] Y Gurevich. The Sequential ASM Thesis. *Bul. EATCS*, 67: 93–124, 1999. Reprinted in *Current Trends in Theoretical Comp. Sc., 2001*, 363–392, World Scientific, 2001.

[13] Yuri Gurevich. Sequential Abstract State Machines capture Sequential Algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.