Study of stepwise simulation between ASM

Patrick Cégielski and Julien Cervelle

LACL – Université Paris-Est Créteil
F-94010 Créteil cedex 2 (France)
{patrick.cegielski,julien.cervelle}@u-pec.fr

Abstract. In this paper we study the notion of stepwise simulation between Abstract State Machines, to explore if some natural change on the original definition would keep it sound. We prove that we have to keep the classical notion and give results about the computability of the simulation itself.

1 Introduction

After one or two centuries of discussion, Richard Dedekind has given a definition of "function": A *function* f from a set A to a set B is a relation $R \subseteq A \times B$ such that if (x, y) and (x, y') belongs to the relation then $y = y'$.

From an informal point of view, a *function* is *computable* if there exists a "mechanical" process which, being given an element x of A, provides the (unique) element $y = f(x)$ of B after some finite "time of computation" (or equivalently some finite number of "steps") if $f(x)$ exists, and runs indefinitely otherwise. A formal definition was given by Alan Turing in 1936, exhibiting a non computable function.

The definition of Turing is universally accepted but other "models of computation" were and are still exhibited for various reasons.

A model of computation is a set of elements, each of them being called a *machine* or a *program*, the denomination depending of the model. Here, we do not define a program as a word over some alphabet because such a pattern is not convenient for ASM. To each machine is associated a finite set of *variables*, each variable taking values in a well defined set. Some variables are used for the *input* and possibly some variables for the *output*; the other variables are called *auxiliary variables*.

Given a finite list v_0, v_1, …, v_n of variables and an associated list D_0, D_1, …, D_n of sets, called *value sets*, an *element of trace* is an assignment for every variable which gives to v_i an assignment in D_i. A *trace* is a finite or infinite sequence of trace elements over the same variables and value sets.

For a given machine M of a given deterministic model of computation $\mathcal{M}$, the *run* of M on a given input is the unique trace whose first element is the initial assignment of the variables: The input variables are initialized with the input and the auxiliary and output variables are initialized as specified by the model.

The element following an element e corresponds to the states of the variables after one step of M starting with the variable values as in the element e.

It is important to note that a run is a trace but the converse is not necessarily true: Indeed, a trace is not necessarily recursive while a run from a constructive model is always recursive.

Let $v_0, v_1, \cdots, v_n$ be a finite list of variables, $v_0, v_1, \cdots, v_p$ be a sublist of input variables, and $D_0, D_1, \cdots, D_n$ be the associated value sets. For a machine M of a model of computation $\mathcal{M}$, the *runs' log* is the set of runs of M for all the possible inputs $\mathcal{I} \in D_0 \times \cdots \times D_p$.

Remark. *A machine is a finite description of a runs' log, which is an infinite set.*

Problem. *Given two models of computation, is a runs' log in one equal to a runs' log in the other?*

The answer to this problem is in general NO but there exists a model of computation, ASM, which has the property that each runs' log of any machine of any model of computation is the runs' log of a machine of the ASM model. Let us make this statement more precise. Yuri GUREVICH has given a schema of languages which is not only a *Turing-complete language* (a language allowing to program each computable function), but which also allows to describe step-by-step the behavior of all algorithms for each computable function (it is an *algorithmically complete language*); this schema of languages was first called *dynamic structures*, then *evolving algebras*, and finally ASM (for *Abstract State Machines*) [2]. In 2000, he proposed the now-called Gurevich's thesis "*the notion of algorithm is entirely captured by the model*" in [3]. A consequence is

For every runs' log, there exists an ASM with that exact runs' log.

ASM is the only known model of computation to have this property. However, authors have considered some other models of computation which are interesting candidates to have a weaker variant of the property. For some of them, it depends on the granularity of a step: Instead of requiring equality of runs, one allows the run of the model to be a specific subsequence of the simulated run. For instance, one keeps only an element every k elements, where the integer k is fixed.

This leads us to the notion of k-*simulation.*

Some authors (see for instance [4]), considering and proving this weaker property for their model, insist on the strict regularity: an element every k elements (for instance 2, 4, 6, ... for $k = 2$) and not just allowing to discard at most $k-1$ elements (for instance 1, 3, 4, 5, 7, ...). This implies adding steps which do nothing (often called "skip" of "nop") in the programs. But these authors give no explanation to justify such a strict constraint.

Then a natural question arises on the behavior of models of computation: Do we need to force the regularity? One way to get enlightenment about the question is to see if we can build two ASM A and B whose traces are all equal up to irregular dilatation but in such a way that the set of points to be removed is, somehow, a non recursive set. This means that the two computations are equivalent but that one of the ASM computes something more in its trace than the other one.

The main result of this paper proves that we have built such a pair of ASM and furthermore that these two ASM are such that one simulates the other but for an again weaker notion: removing at most one point to each trace is sufficient to get the simulation. The result states that the point is not computable given the input.

The paper is organized as follows. The next section gives insight about what is a trace of the execution in the case of a general model of computation and introduces formally ASM and their traces. Section 3 analyzes some possible definitions of equivalent traces and how they are related to computability. It also includes the proofs of announced results.

2 Definitions

2.1 Traces in a general setting of model of computation

Let $\mathcal{M}$ be a (discrete time) model of computation and M an instance of $\mathcal{M}$. We suppose that $\mathcal{M}$ is such that the "state" of M at some time is entirely described by values stored in a finite number of *variables*. Let $v_0, \ldots, v_n$ be these variables. For all i, the variable v_i takes its values from the set D_i (in a more general setting, it is sufficient for D_i to be just a class).

Definition 1. *A* element of trace *for M is an element of $D_0 \times \cdots \times D_n$.*

A trace *is a sequence of elements of trace indexed by $\mathbb{N}$ or by a finite interval $I_k = \{0, \ldots, k\}$ of $\mathbb{N}$. In case of a finite interval I_k, we call k the* length *of the trace.*

For a given M, some variables are distinguished and called input variables*. Without loss of generality, we suppose that $v_0, \ldots, v_k$ for some $k \leq n$ (i.e. the first $k+1$ ones) are the input variables. An* input *for M is an element of $D_0 \times \cdots \times D_k$.*

A run *on input $\imath$ for machine M is the trace $(t_i)_{i \in I}$ where*

- *The trace element t_0 is* initialized *with $\imath$ i.e. the input variables are set as in $\imath$ and the remaining variables are set depending on the definition of $\mathcal{M}$ (see Remark 1).*
- *For all $i \in I$, applying one step of M to t_i leads to t_{i+1} unless M halts and in this case the interval I is $\{0, \ldots, i\}$.*

Remark 1. We suppose that the values of variables v_i for $k < i \leq n$ in t_0 are either fixed by the definition of M or can take any values and in this case, they must have no incidence on the computation of M.

2.2 Definition of ASM

We first introduce ASM, making precise our point of view on ASM, because several definitions exist.

ASM were defined formally in [2]. For this paper, we choose to use only ASM in some normal form (see [1]). We refer the reader to the aforementioned references for the general ASM definition; we just give the formal definition of ASM we use, a variant which is simpler though more verbose and equivalent in power.

Syntax

Definition 2. *An* ASM vocabulary*, or* signature*, is a first-order signature $\mathcal{L}$ with a finite number of static function symbols, a finite number of dynamic function symbols, a finite number of predicate symbols (among which the two boolean constant symbols* `true` *and* `false`*), and an additional symbol, of arity* 0*, (denoted by* `undef`*), logical connectives ($\neg$, $\wedge$, and $\vee$), and the* equality *predicate denoted by* $=$.

Terms of $\mathcal{L}$ are defined by:

- if c is a nullary function symbol (a constant, dynamic or static) of $\mathcal{L}$, then c is a term,
- if $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol (dynamic or static) of $\mathcal{L}$ then $f(t_1, \ldots, t_n)$ is a term.

Definition 3. Boolean terms *of $\mathcal{L}$ are defined inductively by:*

- *if p is an n-ary predicate and $t_1, \ldots, t_n$ are terms of $\mathcal{L}$ then $p(t_1, \ldots, t_n)$ is a boolean term;*
- *if t and t' are terms of $\mathcal{L}$, then $t = t'$ is a boolean term;*
- *if F, F' are boolean terms of $\mathcal{L}$, then $\neg F$, $F \wedge F'$, $F \vee F'$ are boolean terms.*

Definition 4. *Let $\mathcal{L}$ be an ASM signature.* ASM rules *are defined inductively as follows:*

- *An* update *is an expression of the form $f(t_1, \ldots, t_n) := t_0$, where f is a n-ary dynamic functional symbol and $t_0, t_1, \ldots, t_n$ are terms of $\mathcal{L}$ (Recall that constants are allowed but variables are disallowed).*
- *If $R_1, \ldots, R_k$ are updates of signature $\mathcal{L}$, where $k \geq 1$, then the expression $R_1 || \cdots || R_k$ is called a* block *and means parallel execution of the updates.*
- *Finally, if R is a block and φ is a boolean term, the ordered pair $\langle \varphi, R \rangle$ is called a* conditional rule *which must be seen as the instruction* `if` φ `then` R*. In this paper, we call φ the* guard *and R the* block *of the conditional rule.*

Definition 5. *Let $\mathcal{L}$ be an ASM signature. A* program *on signature $\mathcal{L}$, or $\mathcal{L}$-*program*, is a finite set of conditional rules of that signature.*

We can now define ASM.

Definition 6. *An ASM A is a tuple $\langle \mathcal{L}, P\rangle$ where $\mathcal{L}$ is an ASM signature and P is an $\mathcal{L}$-program. We denote by $A_{\mathfrak{D}}$ the set of dynamic symbols of A.*

Semantics

Definition 7. *Let $\mathcal{L}$ be an ASM signature. An ASM* abstract state*, or more precisely an $\mathcal{L}$-*state*, is a synonym for a first-order structure $\mathcal{A}$ of signature $\mathcal{L}$ (an $\mathcal{L}$-structure). We denote by $\overline{t}^{\mathcal{A}}$ the value of the term t in the structure $\mathcal{A}$.*

The universe of $\mathcal{A}$, denoted by $A^{\perp}$, consists of the elements of the *data set A* and a special value $\perp$ (supposedly not in A). The interpretation of the symbol `undef` in $\mathcal{A}^{\perp}$ is always $\perp$.

Definition 8. *Let $\mathcal{L}$ be an ASM signature and A a nonempty set. A* set of modifications *(more precisely an $(\mathcal{L}, A)$-*modification set*) is any finite set of triples $(f, \overline{a}, a)$, where f is an n-ary function symbol of $\mathcal{L}$, $\overline{a} = (a_1, \ldots, a_n)$ is an n-tuple of $A^{\perp}$, and a is an element of $A^{\perp}$.*

Definition 9. *Let $\mathcal{L}$ be an ASM signature, let $\mathcal{A}$ be an $\mathcal{L}$-state and let Π be an $\mathcal{L}$-program. Let $\Delta_{\Pi}(\mathcal{A})$ denote the set defined by as follows:*

1. If u is the update rule $f(t_1, \ldots, t_n) := t_0$ then, denoting $\overline{t_0}^{\mathcal{A}}$ by a_0 , ..., $\overline{t_n}^{\mathcal{A}}$ by a_n, then:

$$\Delta_u(\mathcal{A}) = \{(f, (a_1, \ldots, a_n), a_0)\} \ .$$

2. If B is the block $R_1 || \cdots || R_k$ then:

$$\Delta_B(\mathcal{A}) = \{\Delta_{R_1}(\mathcal{A}), \ldots, \Delta_{R_n}(\mathcal{A})\} \ .$$

3. If T is the conditional rule $\langle \varphi, R\rangle$, we first have to evaluate the expression $t = \overline{\varphi}^{\mathcal{A}}$. We define:

$$\Delta_T(\mathcal{A}) = \begin{cases} \emptyset & \text{if } t \text{ is false,} \\ \Delta_R(\mathcal{A}) & \text{otherwise.} \end{cases}$$

4. Finally, if Π is a program consisting in rules $T_1, \ldots, T_n$, then we define:

$$\Delta_{\Pi}(\mathcal{A}) = \bigcup_{i=1}^{n} \Delta_{T_i}(\mathcal{A}) \ .$$

We defined $\Delta_{\Pi}(\mathcal{A})$ as an $(\Pi, \mathcal{L}, A)$-set of modifications.

Definition 10. *A set of modifications is* incoherent *if it contains two elements $(f, \overline{a}, a)$ and $(f, \overline{a}, b)$ with $a \neq b$. It is* coherent *otherwise.*

Definition 11. *Let $\mathcal{L}$ be an ASM signature, Π an $\mathcal{L}$-program, and $\mathcal{A}$ an $\mathcal{L}$-state.*

The machine's definition must ensure $\Delta_\Pi(\mathcal{A})$ is coherent (otherwise, the machine's definition is invalid). The transform *$\tau_\Pi(\mathcal{A})$ of $\mathcal{A}$ by Π is the $\mathcal{L}$-structure $\mathcal{B}$ defined by:*

- *the base set of $\mathcal{B}$ is the base set A of $\mathcal{A}$;*
- *for any n-ary element f of $\mathcal{L}$ and any element $\overline{a} = (a_1, \ldots, a_n)$ of A^n:*
  - *If there exists some (unique) a such that $(f, \overline{a}, a) \in \Delta_\Pi(\mathcal{A})$, then: $[f]^\mathcal{B}(\overline{a}) = a$.*
  - *Otherwise: $[f]^\mathcal{B}(\overline{a}) = [f]^\mathcal{A}(\overline{a})$.*

Definition 12. *Let $\mathcal{L}$ be an ASM signature, Π an $\mathcal{L}$-program, and $\mathcal{A}$ an $\mathcal{L}$-state. The* computation *is the sequence of $\mathcal{L}$-states $(\mathcal{A}_n)_{n \in \mathbb{N}}$ defined by:*

- *$\mathcal{A}_0 = \mathcal{A}$ (called the* initial algebra *of the computation);*
- *$\mathcal{A}_{n+1} = \tau_\Pi(\mathcal{A}_n)$ for $n \in \mathbb{N}$.*

For ASM, a computation halts *if there exists a fixed point $\mathcal{A}_{n+1} = \mathcal{A}_n$. In this case, this fixed point $\mathcal{A}_n$ is the* result *of the computation.*

Definition 13. *The formal* semantics *is the partial class function which transforms $\langle \Pi, \mathcal{A} \rangle$, where Π is an ASM program and $\mathcal{A}$ a state, in the fixed point $\mathcal{A}_n$ obtained by iterating τ_Π starting from $\tau_\Pi(\mathcal{A})$ until a fixed point is reached if such a fixed point exists, otherwise it is undefined.*

Finally, though it is not directly mentioned in the original paper defining ASM, as we need to deal with simulation, we need a formal definition of how input is treated.

For some ASM signature $\mathcal{L}$, some of the dynamic symbols will be used for the input. We call them *input* symbols. To construct the initial algebra $\mathcal{I}(e)$ of an ASM computation on input e, we use the following:

- Some set E to use $E^\perp$ as universe for the algebra.
- For all static variables of arity n of $\mathcal{L}$, we assign a function from $(E^\perp)^n \to E$.
- The input is stored in input symbols (so input can be some values of $E^\perp$ or functional).
- The rest of the dynamic symbols are set to the constant function equals to $\perp$.
- The set E is called the *data set* of the ASM.
- We say that an ASM is m-ary when it has only one input symbol of arity m.

2.3 Trace

We restate here the definition of trace introduced in Section 2.1 in the special case of ASM.

Definition 14. *For some ASM, a* trace element *is the values of all the dynamic symbols (input or not).*

The trace *of some ASM A on input e is the sequence of elements of trace $(t_i)_{i\in I}$ where I is either $\mathbb{N}$ or $\{0,\ldots,\ell\}$ for some ℓ where t_i is the restriction of $\mathcal{A}_i$ to dynamic symbols if $(\mathcal{A}_n)_{n\in\mathbb{N}}$ is the computation of the ASM starting from $\mathcal{I}(e)$. Moreover, if the computation does not halt, then $I = \mathbb{N}$, and otherwise, $I = \{0,\ldots,\ell\}$ when ℓ is the number of steps before halting (that is the smallest with $\tau_\Pi(\mathcal{A}_\ell) = \mathcal{A}_\ell$). We denote I by $\mathrm{dom}(t)$.*

If y is a trace element and A is a subset of the dynamic symbols, $y \restriction A$ is the trace element which assign values to dynamic symbols of A as in y. If t is a trace, $t \restriction A$ is the trace $((t_i \restriction A)_i)_{i\in\mathrm{dom}(t)}$.

2.4 Self-detection of a halting ASM

It is possible (well-known result of the folklore) in any ASM to write a boolean expression (subsequently called `HasHalted`) which evaluates true when it is in a state which is a fixed point. The idea is to test if all assignments do not change the assigned value, checking also if the set of modifications is coherent.

3 Analysis

We first introduce a preliminary notion of subtrace.

Definition 15. *Let t_1 and t_2 be two traces. We say that t_2 is a* subtrace *of t_1 if there exists a strictly increasing function $s : \mathrm{dom}(t_1) \to \mathrm{dom}(t_2)$ such that:*

1. *$\forall i \in \mathrm{dom}(t_1), t_1(i) = t_2(s(i))$,*
2. *$s(0) = 0$*
3. *$\mathrm{dom}(t_1)$ is finite if and only if $\mathrm{dom}(t_2)$ is finite*
4. *if $\mathrm{dom}(t_1)$ is finite, then $s(\max \mathrm{dom}(t_1)) = \max \mathrm{dom}(t_2)$*

The function s and condition 1. ensure that t_1 is extracted from the elements of t_2 keeping the order. The next conditions state that the two traces have the same start (2.), both have an end or both are infinite (3.) and in the finite case, have the same end (4.).

We define the notion of k-weak regular subtracing which is a more constrained notion of subsequence.

Definition 16 (Weak regular subtrace). *Let t_1 and t_2 be two traces and k some positive integer. We say that t_2 is a k-*weak regular subtrace *of t_1 if t_2 is a subtrace of t_1 and furthermore if $\forall i \in \mathrm{dom}(t_1)$, $k \geq s_1(i+1) - s_1(i) > 0$.*

The last condition ensures that to build the subtrace of t_2, one cannot skip more than $k-1$ elements: two consecutive terms taken from t_2 always belong to some windows of length k.

We say that s is a guide *of the k-weak regular subtracing of t_1 by t_2.*

Finally, we say that the set $w = \mathrm{dom}(t_2) \setminus \{s(i) \mid i \in \mathrm{dom}(t_1)\}$ is a witness *of the k-weak regular subtracing of t_1 by t_2.*

Remark 2. Witnesses and guides are not unique : if the same trace element x occurs twice in t_2 ($t_2(a) = t_2(b) = x$) and corresponds to only one trace element in t_1 ($t_1(c) = x$), the function s can possibly choose either ($s(c) = a$ or $s(c) = b$) provided both choices comply to the weak regular subtracing condition.

Remark 3. Note that guides and witnesses are linked and uniquely defined and computable one from another. Indeed, the guide is the increasing enumeration of the complement of the witness.

For instance, consider the trace $t_1 = [x = 0], [x = 2], [x = 4], \ldots$ (x is incremented by 2 at each step) and the trace $t_2 = [x = 0], [x = 1], [x = 2], \ldots$ A (x is incremented by 1 at each step). The only guide of the 2-weak regular subtracing of t_1 by t_2 is $s : t \mapsto 2t$. The only witness is $\{2k + 1 \mid k \in \mathbb{N}\}$.

We now define the stronger notion of k-regular subtrace.

Definition 17 (Regular subtrace). *Let t_1 and t_2 be two traces (finite or infinite) and k be some integer. We say that t_2 is a k-*regular subtrace *of t_1 if t_2 is a subtrace of t_1 and furthermore if $\forall i \in \mathrm{dom}(t_1)$, $s(i) = ki$.*

For regular subtracing, the chosen elements are picked up one every exactly k elements.

Remark 4. If t_2 is a k-regular subtrace of t_1, it is also a k-weak regular subtrace of t_1.

Notation. For A an ASM and e some input, we denote by t_e^A the run of A on initial algebra $\mathcal{I}(e)$.

When the ASM is denoted A_i, we denote by t_e^i the run $t_e^{A_i}$.

Definition 18 (Weak simulation). *Some ASM B k-*weakly simulates *some other A if $A_{\mathcal{D}} \subseteq B_{\mathcal{D}}$ and for all input e, the run t_e^A is a k-weak regular subtrace of $t_e^B \restriction A$. We denote by $\mathcal{W}_k^e(A, B)$ the set of witnesses for this weak regular subtracing and by $\mathcal{G}_k^e(A, B)$ the set of guides.*

Definition 19. *An ASM A is* arithmetic *if its data set is $\mathbb{N}$ and all the static functions are Turing computable.*

Theorem 1. *Let A and B be two arithmetic ASM such that B k-weakly simulates A. Then, given any computable input e on this input, there is a computable element in $\mathcal{G}_k^e(A, B)$.*

Proof. If both traces are finite, the guides of $\mathcal{G}_k^e(A, B)$ are all finite and therefore computable. Without loss of generality, we may assume in the sequel that both traces are infinite.

Firstly, note that one can simulate the execution of A and B on input e. This can be done since the modifications of the dynamic values only concern a finite number of elements of the domains which are integers. More precisely, for all dynamic symbols f of domain D (D is $\mathbb{N}^\ell$ where ℓ is the arity of f), the simulator saves a table T_f from $\mathfrak{P}(D \times \mathbb{N})$. The simulator evaluates the program's guard with a call by value scheme: when the simulator is computing the value $f(x)$ for some x in D, the simulator first checks if there exists y such that $\langle x, y \rangle \in T_f$. In this case, the result is y. Otherwise, the result is taken from e for input symbols and $\perp$ for other dynamic symbols. When $f(x)$ is assigned a value v, the simulator adds $\langle x, v \rangle$ in T_f possibly removing an ancient value associated to x in the table. With this way of representing the state of an arithmetic ASM, it is decidable whether two trace elements corresponding to the same ASM are equal. We denote by $[q]$ the minimal representation of the state q using such tables (minimal meaning removing entries $\langle x, \perp \rangle$ for dynamic symbols and entries $\langle x, y \rangle$ where y is the same value as in e from input symbols).

We conclude that the functions t_1 and t_2 defined as $t_1(i) = [t_e^A(i)]$ and $t_2(i) = [t_e^B \upharpoonright A_{\mathcal{D}}(i)]$ are computable. Moreover, by definition of an ASM, the state of an ASM after one step only depends on the current state. Therefore, there is a computable function a such that $t_1(i+1) = a(t_1(i))$.

We now construct an enumerable rooted DAG (directed acyclic graph) D whose nodes are labeled by integers. The DAG D is built such that, for all paths from the root p, the sequence $(x_i)_{i \in \mathbb{N}}$, where x_i is the integer labelling p_i, is a guide of $\mathcal{G}_k^e(A, B)$.

The DAG D is built inductively. We firstly label the root by 0. For each node n labeled by i, we look consider the possibly empty set $S_i = \{i' \mid 1 \leq i' \leq k \wedge t_2(i') = a(t_2(i))\}$. For all $i' \in S_i$, we add as a child to n labeled by i'. If two nodes have the same label, they are merged. Since the label is only increasing, at some point, the algorithm performing the construction knows that all the ancestors of a given node have been produced.

Let us prove by induction that paths from the root are labeled by guides. The initialization comes from the fact that $t_e^A(0) = t_e^B \upharpoonright A_{\mathcal{D}}(0)$. For the induction step, consider the first n elements of a path of D: $x_0, \ldots, x_{n-1}$ and such that $\forall i < n$, $t_e^A(i) = t_e^B \upharpoonright A_{\mathcal{D}}(x_i)$. For any child x_n of x_{n+1}, one has $[t_e^A(n)] = t_1(n) = a(t_1(n-1)) = a(t_2(x_{n-1})) = t_2(x_n) = [t_e^B \upharpoonright A_{\mathcal{D}}(x_n)]$. Therefore, $t_e^A(n) = t_e^B \upharpoonright A_{\mathcal{D}}(x_n)$.

In order to end the proof, we need to show that D has a computable infinite path. Firstly, we consider D', subgraph of D, which is a tree, keeping, for each node, only the edge to its smaller labeled ancestor. By construction of D, each path of D', labeled by x, is such that $\forall i$, $|x_{i+1} - x_i| \leq k$.

Let us show, by contradiction, that there are at most k infinite paths. Suppose that there are $k+1$ infinite paths labeled by $x^{(0)}, \ldots, x^{(k)}$. Let n be such that the paths have no common node labeled by an integer greater then n. Since $\forall i, j$, $|x_{i+1}^{(j)} - x_i^{(j)}| \leq k$, it means that the set $\{n+1, \ldots, n+k\}$ of cardinality k intersects the $k+1$ sequences $x^{(0)}, \ldots, x^{(k)}$ which contradicts the definition of n. Hence, D' is a computable tree with at most k infinite paths. Then the sequence of labels of each of these paths are computable. Indeed, to enumerate them increasingly: the programs knows the (finite) beginning of the path up to the first node q

labeled by an integer greater than n. From node q, the tree has only one infinite branch. Then to enumerate the successor of some node r, the program starts in parallel searches in D' from all the children of r. All of the children but one, r', belong to finite subtrees. Once they are all found (i.e. all searches have halted but one), the program output r' and continues the enumeration from r'.

Proposition 1. *There exist* 0*-ary ASM A and B such that B 2-simulates A and such that no element of $\mathcal{W}_2^{\emptyset}(A,B)$ is recursive.*

Proof. Let `A` be the ASM, with the successor function as static symbol and dynamic 0-ary symbols `m` and `n` with program:

```
if (n = undef)
  m := 0
  n := 1
if (n ≠ undef)
  n := n+1
```

This ASM simply keeps `m=0` and increments `n` from 1 to infinity.

Let f be the characteristic function of a non-recursive set (that is a total function from $\mathbb{N}$ to $\{0,1\}$ such that $f^{-1}(1)$ is non-recursive). Let `B` be the ASM, with f, the division by 2, parity test, and the integer successor as static symbols and dynamic 0-ary symbols `m` and `n`, with program:

```
if (n = undef)
  m := 0
  n := 1
if (n is even ∧ f(n/2) = 1)
  m := 1
if (n is odd ∨ m = 1)
  m := 0
  n := n+1
```

This ASM does what A does but delays the incrementation of `n` when $f(n/2) = 1$. The ASM B 2-weakly simulates A by simply omitting the steps where $f(n/2) = 1$. These steps occurs at time t where $\exists k \neq 0$, such that $f(k) = 1$ and $t = \sum_{i=1}^{k} 2 + \delta_{f(i)1}$ (where δ is the Kronecker delta). Hence the only witness for B 2-weak simulating A is $W = \{\sum_{i=1}^{k} 2 + \delta_{f(i)1} \mid k \in \mathbb{N} \setminus \{0\} \wedge f(k) = 1\}$. Since $f^{-1}(1)$ is a non-recursive set and can be computed from W, we conclude that W is non-recursive.

Theorem 2. *There exist some arithmetic ASM A and B such that:*

- *The ASM B 2-weakly simulates A and the set $\mathcal{W}_2(A,B)$ contains only finite sets.*
- *The set $\mathcal{W}_2(A,B)$ is non-recursive.*

Proof. Let $E \subset \mathbb{N}$ be a recursively enumerable, non-recursive set. Let f be a recursive function such that $E = \{x \mid f(x) \text{ halts}\}$. Let F be an arithmetic ASM computing f. Without loss of generality, assume that the input for F is in the variable `x`.

We design two ASM A and B as follows.

- The ASM A and B have all the symbols of F.
- We add to A and B the new variables `s`, `c`, `d`, and `m`, all initialized to 0. The purpose of these variables is as follows:
  - `s` is used to have an initial step which copies `x` into `c`.
  - `c` is a counter which, after the initial step, decreases from `x` to 0 and afterwards remains equal to 0.
  - `d` is a counter which increases from 0 after `c` has reached 0. It increases while the simulation of A has not halted.
  - `m` is turned to 1 for one step at the beginning and at the end of the simulation of F but only in B. Otherwise it remains equal to 0. It is also the case during the whole run in A.
- To allow the expected behavior of A and B, all rules $\langle g, R\rangle$ of A are put into A and B as $\langle g \wedge \mathtt{s} = 1 \wedge \mathtt{c} = 0, R\rangle$.
- In A, we add the three following rules:

```
if (s = undef)
  c := x
  s := 1
if(s = 1 ∧ c > 0)
  c := c-1
if(c = 0 ∧ ¬HasHalted)
  d := d+1
```

- In B, we do almost the same but, using the variable `m`, we add differences in the execution of the ASM (`HasHalted` is the halt time detection expression for ASM F):

```
if(s = undef)
  c := x
  s := 1
if(s = 1 ∧ c > 0)
  c := c-1
if(s = 1 ∧ c = 0 ∧ m = undef)
  m := 1
if(s = 1 ∧ c = 0 ∧ m = 1)
  m := undef
if (c = 0 ∧ ¬HasHalted)
  d := d+1
if (HasHalted)
  m := 1
```

The runs for A and B on input x start with:

$$\imath + [\mathtt{s} = \bot, \mathtt{x} = x, \mathtt{c} = \bot, \mathtt{m} = \bot], (\imath + [\mathtt{m} = \bot, \mathtt{s} = 1, \mathtt{x} = x, \mathtt{c} = i])_{i \text{ from } x \text{ to } 0}$$

where $\imath$ contains the initial values of all other variables.

Then, the next element of the run for B, is

$$\imath + [\mathtt{m} = 1, \mathtt{s} = 1, \mathtt{x} = x, \mathtt{c} = 0]$$

Afterwards, the elements of the runs for A and B are again identical and contain the computation of $f(x)$ by F.

If the computation of $f(x)$ does not halt, the only witness of $\mathcal{W}_2^x(A, B)$ is the singleton $\{x\}$.

If the computation of $f(x)$ halts, the run of A ends after the computation as does the run of B which has one more element, of index r_x, equal to $\jmath + [\mathtt{m} = 1]$ where $\jmath$ is the state of both A and B at the previous step. In this case, the only witness of $\mathcal{W}_2^x(A, B)$ is $\{x, r_x\}$.

We conclude that $W = \{\mathcal{W}_k^e(A, B) \mid e \text{ input}\} = \{\{x\} \mid f(x) \text{ does not halts}\} \cup \{\{x, r_x\} \mid f(x) \text{ halts}\}$. If one can enumerate W, one can enumerate $\mathbb{N} \setminus E$ and therefore, being enumerable, E is computable which is a contradiction. We conclude that W is not enumerable and consequently non-recursive.

Conclusion

In this paper, we studied the simulation of computation models by ASM where one step of simulation is executed by several steps of the ASM. We showed that the correct simulation must ensure that each simulated step corresponds to the same number of steps of the ASM. Indeed, if the number can be variable, the simulated model could be computing more information, not directly in its variables but looking at which steps the content of some variables is modified.

References

1. Cégielski, Patrick and Guessarian, Irène, Normalization of Some Extended Abstract State Machines, Fields of Logic and Computation, Blass, Andreas and Dershowitz, Nachum and Reisig, Wolfgang, eds, 165–180, Springer-Verlag, 2010
2. Gurevich, Yuri, Reconsidering Turing's Thesis: Toward More Realistic Semantics of Programs, University of Michigan, Technical Report CRL–TR–38–84, EECS Department (1984)
3. Gurevich, Yuri, A New Thesis, Abstracts, American Mathematical Society, p. 317 (August 1985)
4. Marquer, Yoann and Valarcher, Pierre, An Imperative Language Characterizing PTIME Algorithms, Studies in Weak Arithmetics 3, Patrick Cégielski, Ali Enayat and Roman Kossak, eds., Lecture Note 217, CSLI Publications, Stanford, 2016.