A Behavioural Theory for Reflective Sequential Algorithms

Flavio Ferrarotti(✉), Klaus-Dieter Schewe, and Loredana Tec

Software Competence Center Hagenberg, Hagenberg, Austria
flavio.ferrarotti@scch.at, kd.schewe@gmail.com, loredana.tec@gmail.com

Abstract. We develop a behavioural theory of reflective sequential algorithms (RSAs), i.e. algorithms that can modify their own behaviour. The theory comprises a set of language-independent postulates characterising the class of RSAs, an abstract machine model that provably satisfies the postulates, and a proof that all RSAs are captured by this machine model. As in Gurevich's thesis for sequential algorithms RSAs are sequential-time, bounded parallel algorithms, where the bound depends on the algorithm only and not on the input. Different from the class of sequential algorithms every state of an RSA includes a representation of the algorithm in that state, thus enabling linguistic reflection. The model of reflective Abstract State Machines (rASMs) extends sequential ASMs using extended states that include an updatable representation of the main ASM rule to be executed by the machine in that state.

1 Introduction

Self-adaptive systems have recently attracted a lot of interest in research, in particular in connection with systems of (cyber-physical) systems [9]. Adaptivity refers to the ability of a system to change its own behaviour. In the context of programming this concept, known under the term *linguistic reflection*, appears already in LISP [11], where programs and data are both represented uniformly as lists, and thus programs represented as data can be executed dynamically by means of an evaluation operator. Run-time and compile-time linguistic reflection in programming and database research has been investigated in general by Stemple, Van den Bussche and others in [12,13].

This raises the questions how the development of adaptive systems can be supported by state-based rigorous methods such Abstract State Machines (ASMs) [5]. These methods are coupled with a genericity promise, i.e. they can be applied universally to a large class of systems supporting rigorous specification on any level of abstraction, seamless step-wise refinement from high-levels of abstraction down to implemented code, validation and tracing of requirements

The research reported in this paper results from the projects *Behavioural Theory and Logics for Distributed Adaptive Systems* and *Higher-Order Logics and Structure* supported by the **Austrian Science Fund (FWF: [P26452-N15] & [I2420-N31])**.

A. K. Petrenko and A. Voronkov (Eds.): PSI 2017, LNCS 10742, pp. 117–131, 2018.
https://doi.org/10.1007/978-3-319-74313-4_10

through the refinement process, and verification of specifications against the requirements on grounds of dedicated logics. However, reflective algorithms are not yet covered.

Gurevich's celebrated *sequential ASM thesis* [8] states that sequential algorithms are captured by sequential ASMs. A key contribution of this thesis is the language-independent characterisation of a reflective algorithm by a small set of intuitively understandable *postulates*, by means of which a definition of a sequential algorithm on an arbitrary level of abstraction is given. Then it can be shown that every sequential algorithm as stipulated by the postulates can be step-by-step simulated by a sequential ASM.

Based on this thesis the notion *behavioural theory* has been introduced for a triplet comprising (1) a set of *postulates* that characterises a class of algorithms or systems, (2) an abstract machine model together with a *plausibility* proof that the abstract machine satisfy the postulates, and (3) a *characterisation* proof that all algorithms stipulated by the postulates are captured by the abstract machine model. That is, the sequential ASM thesis provides the behavioural theory of sequential algorithms. Other examples cover the behavioural theory of parallel algorithms developed by Blass and Gurevich [1,2], its simplification by Ferrarotti et al. [6] using a different set of postulates, the behavioural theory of concurrent algorithms [4], and the behavioural theory of non-deterministic database transformations [10].

In this paper we investigate a behavioural theory for reflective, sequential algorithms, which was conjectured in [7]. In light of the significantly increased technical difficulties that have to be addressed when unbounded parallelism is permitted (compare the proofs in [6] with those in the sequential ASM thesis [8]) we first restrict the emphasis on sequential algorithms, where parallelism is a priori bound and does not depend on the state.

We first develop a set of postulates characterising *reflective sequential algorithms* (RSAs). The key issue is that an RSA must have some representation of itself, but this has to be left completely abstract. We argue that this is possible, which leads to extended states, where abstract terms that appear in the description of the algorithm are used as values, which requires a distinction concerning their interpretation in a state. The tricky problem is the generalisation of bounded exploration, for it is clear that all means of an algorithm to change itself must appear somehow in the algorithm's description. We argue that there is still a bounded exploration witness, i.e. a set of ground terms that determines the update sets yielded in a state, but the bounded exploration postulate will nonetheless require some sophisticated differentiation concerning the interpretation of terms. The postulates for RSAs will be discussed in Sect. 2.

In Sect. 3 we proceed with the definition of reflective sequential ASMs (rASMs), which will be a straightforward extension of ASMs using a dedicated location *self* capturing the (syntax of the) sequential ASM that is to be applied in this state. This determines the runs of a rASM with the difference that in each step now a possibly different ASM may have been used to determine the

updates that mark the state changes. We also briefly sketch the plausibility theorem though it commonly addresses the simpler proof direction.

Section 4 addresses the proof of the characterisation theorem, which is again accomplished by a sequence of lemmata, the key problem being that there is a theoretically unbounded number of different algorithms that nonetheless have to be handled uniformly. This section will be the technical key contribution of this paper. We conclude with a brief summary and outlook in Sect. 5.

2 Reflective Algorithms and Their Axiomatisation

The celebrated sequential ASM thesis needs only three simple, intuitive postulates to define sequential algorithms (for details see the deep discussion in [8]):

Sequential time: Each sequential computation proceeds by means of a *transition function* $\tau : \mathcal{S} \to \mathcal{S}$, which maps a state $S \in \mathcal{S}$ to its successor state $\tau(S)$.

Abstract state: Each state $S \in \mathcal{S}$ is a *Tarski structure* defined over a signature Σ, i.e. a set of function symbols, by means of interpretation in a base set B_S. States, initial states and transitions are closed under isomorphisms.

Bounded exploration: There is a fixed, finite set of ground terms W called *bounded exploration witness* such that whenever two states coincide on W, the update sets that determine the changes in the transition to the respective successor states are equal.

The postulates imply that sequential algorithms can only check agreement between states on a *fixed* and *finite* set of *ground terms* (i.e., the bounded exploration witness in the bounded exploration postulate for sequential algorithms). Reflective algorithms, however, do not satisfy this principle, as the following simple Example 1 shows. The RSA in the example does *not* satisfy the bounded exploration postulate for sequential algorithms. However, it is NOT the question, whether a different, non-reflective algorithm exists that solves the same problem, but whether such an algorithm would also be *behaviourally equivalent*.

Example 1. We describe a RSA that takes as input a search term t, a perfect binary tree T, i.e., a binary tree in which all interior nodes have two children and all leaves have the same depth, and a function *label* which maps the set of nodes to an arbitrary set of labels. It traverses the graph in a breadth first order starting by its root r. If the term t appears as label of some node in the input tree, then the algorithm updates *result* to the lowest level in the tree which contains a node labeled with t.

We assume that in every initial state $level = 0$, $currentNode = r$ and $result = undef$. Let *cond* be the following function from the natural numbers to Boolean terms:

$$
\begin{aligned}
cond(0) &::= label(r) = t \\
cond(1) &::= label(leftChild(r)) = t \vee label(rightChild(r)) = t \\
cond(2) &::= label(leftChild(leftChild(r))) = t \vee label(rightChild(leftChild(r))) = t \vee \\
&\quad\ label(leftChild(rightChild(r))) = t \vee label(rightChild(rightChild(r))) = t \\
&\ \ \vdots \qquad\qquad \vdots \\
cond(n) &::= \underbrace{label(leftChild(\ldots leftChild(r)\ldots)) = t \vee \cdots}_{} \\
&\qquad\qquad \underbrace{\cdots \vee label(rightChild(\ldots rightChild(r)\ldots)) = t}_{2^n}.
\end{aligned}
$$

The algorithm works as follows:

```
1: if currentNode ≠ undef ∧ result = undef then
2:     if label(r) = t then
3:         result := level
4:     else
5:         currentNode := leftChild(currentNode)
6:         level := level + 1
7:         Replace the Boolean term in the if-statement in line 2 by the
               interpretation of cond(level+1)
8:     endif
9: endif
```

Notice that during a run or computation of this algorithm not only the state of the algorithm evolves, but also the algorithm itself. In fact, the Boolean term in the if-statement in line 2 changes with every state transition until either the searched term t is found or all the levels of the input tree have been exhausted.

As we consider input trees of arbitrary size, this means that there is no fixed and finite bounded exploration witness for this algorithm, as we would need to either include all Boolean terms in the infinite set $\{cond(0), cond(1), \ldots\}$, or include a different Boolean term $cond(level + 1)$ depending on the interpretation of $level + 1$ in the current state. □

2.1 Reflective Sequential Time Postulate

Clearly, when extending the notion of sequential algorithm to include reflection we think of pairs (S_i, P_j) comprising a state S_i (as in the sequential thesis), and a sequential algorithm P_j. Thus, we can consider transition functions $\tau_j : (S_i, P_j) \mapsto (S_{i+1}, P_j)$ without changing the sequential algorithm P_j. Likewise we may consider transition functions $\sigma_i : (S_i, P_j) \mapsto (S_i, P_{j+1})$ changing only the algorithm. In general, a transition of a RSA can then involve both: updates to the state and updates to the algorithm.

Postulate 1 (Reflective Sequential Time Postulate). Let $\mathcal{S}_P$ and $\mathcal{I}_P$ denote the set of states and initial states of a sequential algorithm P, respectively ($\mathcal{I}_P \subseteq \mathcal{S}_P$). A RSA $\mathcal{A}$ consists of the following:

- A non-empty set $\mathcal{P}_\mathcal{A}$ of *sequential algorithms*;
- An *initial algorithm* $P_0 \in \mathcal{P}_\mathcal{A}$;
- A non-empty set $\mathcal{S}_\mathcal{A} = \bigcup_{P_i \in \mathcal{P}_\mathcal{A}} \mathcal{S}_{P_i}$ of *states*;
- A non-empty set $\mathcal{I}_\mathcal{A} = \mathcal{I}_{P_0} \subseteq \mathcal{S}_\mathcal{A}$ of *initial states*;
- A set of extended-states $\mathcal{E}_\mathcal{A} = \mathcal{S}_\mathcal{A} \times \mathcal{P}_\mathcal{A}$;
- A *one-step transformation* function $\tau_\mathcal{A} : \mathcal{E}_\mathcal{A} \to \mathcal{E}_\mathcal{A}$ such that $\tau_\mathcal{A}((S, P)) = (S', P')$ only if $\tau_P(S) = S'$ for the one-step transformation function τ_P of the sequential algorithm P.

Then a *run* or *computation* of a reflective algorithm corresponds to a sequence of pairs (S_0, P_0), (S_1, P_1), (S_2, P_2), ..., where S_0 is an initial state in $\mathcal{I}_\mathcal{A}$, P_0 is the initial algorithm, and $(S_{i+1}, P_{i+1}) = \tau_\mathcal{A}((S_i, P_i))$ holds for every $i \geq 0$.

This leads to the following three fundamental questions which we try to answer in the remaining part of this section:

Q1. How can we finitely represent a sequential algorithm P without having to adopt a concrete language for the specification of P?
Q2. How can we finitely characterise changes to the representation of the sequential algorithms in all states?
Q3. How can we define behavioural equivalence of RSAs independently from the representation of the sequential algorithms in each state?

2.2 Reflective Abstract Extended-State Postulate

Concerning Q1 we observe that according to the sequential ASM thesis it suffices to represent a sequential algorithm P by a set of pairs $(S, \Delta(P, S))$ comprising a state S and the update set of P in that state. A consequence of the proof of the sequential ASM thesis in [8] is that update sets $\Delta(P, S_i)$ $(i = 1, 2)$ are equal, if the states S_1 and S_2 are W-equivalent for a fixed bounded exploration witness W. We have $S_1 \sim_W S_2$ iff $E_{S_1} = E_{S_2}$, where E_S is the equivalence relation on W defined by $E_S(t_1, t_2) \equiv val_S(t_1) = val_S(t_2)$[1]. It is therefore sufficient to replace the state S by a condition $\varphi_{[S]}$, which evaluates to true on states that are W-equivalent to S. As there can only be finitely many W-equivalence classes, we obtain an abstract finite representation by a finite set of pairs (φ_i, Δ_i) $(i = 1, \ldots, k)$.

Therefore, we conclude that we can capture the state-algorithm pairs in a RSA by an extension Σ_{ext} of the signature Σ using additional function symbols to represent the sequential algorithm, e.g. capturing in the state the signature of the algorithm as well as some syntactic description of it. For this, we must further permit new function symbols to be created, which can be done by exploiting the concept of "reserve". We also conclude that the representation of algorithms in a state requires terms that are used by the algorithms to appear as values. So we have to allow terms over Σ (including the dormant function symbols in the reserve) to be at the same time values in an extended base set. In order

[1] As usual, $val_S(t)$ denotes the interpretation of a ground term t as a value in the base set of a state S.

to distinguish the interpretation of such terms t as values $val_{(S,P)}(t)$ of the base set of an extended-state (S, P) (which in any extended-state evaluate to themselves) from their interpretation as terms over Σ, we use $raise_{(S,P)}(t)$ to denote the latter case. We may further assume that *raise* results in a proper term over Σ not containing any extra-logical constructs that are needed in the representation of an algorithm such as keywords.

Postulate 2 (Reflective Abstract Extended-State Postulate). Let $\mathcal{A}$ be RSA. Fix a finite signature Σ_{algo} of function symbols so that every algorithm $P \in \mathcal{P}_{\mathcal{A}}$ can be finitely represented as some first-order structure of signature Σ_{algo}.

- Every P in $\mathcal{P}_{\mathcal{A}}$ is a first-order structure of signature Σ_{algo} which encodes a finite representation of a sequential algorithm.
- Every state S in $\mathcal{S}_{\mathcal{A}}$ is a first-order structure of some signature Σ_S such that $\Sigma_S \cap \Sigma_{algo} = \emptyset$.
- Every extended-state (S, P) in $\mathcal{E}_{\mathcal{A}}$ is a first-order structure of (extended) signature $\Sigma_{ext} = \Sigma_S \cup \Sigma_{algo}$.
- The one-step transformation function $\tau_{\mathcal{A}}$ does not change the base set of any extended-state of $\mathcal{A}$.
- The sets $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{I}_{\mathcal{A}}$ are closed under isomorphisms.
- If $(S_1, P_1), (S_2, P_2) \in \mathcal{E}_{\mathcal{A}}$, S_1 and S_2 are isomorphic, P_1 and P_2 are behavioural equivalent sequential algorithms[2], and further $\tau_{\mathcal{A}}(S_1, P_1) = (S_1', P_1')$ and $\tau_{\mathcal{A}}(S_2, P_2) = (S_2', P_2')$, then also S_1' and S_2' are isomorphic and P_1' and P_2' are behavioral equivalent.

Same as in the sequential ASM thesis, we need some minimal *background of computation*. Therefore, for every extended state (S, P), we assume that S includes a binary function "=" for equality, nullary functions `true`, `false` and `undef` with `true` $\neq$ `false` and `true` $\neq$ `undef`, the usual Boolean functions, the set of all ordered pairs, and an infinite reserve of elements. As explained before, we further assume that P_i includes, as values in its base set, the set of all possible ground terms (including the dormant function symbols in the reserve).

2.3 Reflective Bounded Exploration Postulate

Concerning Q2 the problem is that in general we must expect that each sequential algorithm P_i represented in an extended-state (S_i, P_i) has its own bounded exploration witness W_i. However, we know from the sequential ASM thesis that W_i is somehow contained in the finite representation of P_i. For instance, the sequential ASM rule constructed in the proof of the sequential ASM thesis only contains subterms of terms in W_i, and this holds analogously for any other representation of P_i. This implies that the terms in W_i result by interpretation from terms that appear in the representation of any sequential algorithm. Thus, there must exist a finite set of terms W such that its interpretation in an extended

[2] Two sequential algorithms P_1 and P_2 are *behavioural equivalent* if $\mathcal{S}_{P_1} = \mathcal{S}_{P_2}$, $\mathcal{I}_{P_1} = \mathcal{I}_{P_2}$ and $\tau_{P_1} = \tau_{P_2}$. Behavioural equivalent sequential algorithms have the same runs.

state yields both values and terms, and the latter represent W_i. We will continue to call W a *bounded exploration witness*. Consequently, the interpretation of W and of its interpretation in an extended state suffice to determine the update set in that state. This leads to our *bounded exploration postulate* for RSAs.

Definition 1 (Strong Coincidence). *Let (S, P) and (S', P') be extended-states of a RSA. Let $W = W_{st} \cup W_{wt}$ be a set of ground terms. We say that (S, P) and (S', P')* strongly coincide *over W iff the following holds:*

- *For every $t \in W_{st}$, $val_{(S,P)}(t) = val_{(S',P')}(t)$.*
- *For every $t \in W_{wt}$,*
  1. *$val_{(S,P)}(t) = val_{(S',P')}(t)$.*
  2. *$val_{(S,P)}(raise_{(S,P)}(t)) = val_{(S',P')}(raise_{(S',P')}(t))$.*

In our third and last postulate we use $\Delta(\mathcal{A}, (S, P))$ to denote the set of updates produced by a RSA $\mathcal{A}$ in an extended-state (S, P).

Postulate 3 (Reflective Bounded Exploration Postulate). For every RSA $\mathcal{A}$, there is a finite set $W = W_{st} \cup W_{wt}$ of ground terms such that $\Delta(\mathcal{A}, (S, P)) = \Delta(\mathcal{A}, (S', P'))$ whenever extended-states (S, P) and (S', P') of $\mathcal{A}$ *strongly coincide* on W.

If a set of ground terms $W = W_{st} \cup W_{wt}$ satisfies the reflective bounded exploration postulate, we call it a *reflective bounded exploration witness* (R-witness for short) for $\mathcal{A}$.

2.4 Reflective Sequential Algorithms and Behavioural Equivalence

Our three postulates give us the following machine independent definition of RSAs.

Definition 2. *A* reflective sequential algorithm *(RSA) is an algorithm satisfying the Reflective Sequential Time, Reflective Abstract State and Reflective Bounded Exploration Postulates.*

Example 2. Let us consider the algorithm in Example 1. The reflective sequential time and reflective abstract state postulates are clearly satisfied by this algorithm. Let

$$W_{st} = \{currentNode, undef, result, level, level + 1, leftChild(currentNode)\}$$

and $W_{wt} = \{cond(level)\}$. It is not difficult to see that if two extended-states coincide on $W = W_{st} \cup W_{wt}$, then the algorithm considered in this example produces the same set of updates in both extended-states. Thus, it also satisfies the reflective bounded exploration postulate, and consequently our definition of RSA. □

Next, we turn our attention to our final fundamental question Q3. The problem here is that the notion of behavioural equivalence of two sequential algorithms is bound to these having the same signature, on grounds of which we can request that the sets of runs must be identical. This cannot be carried over to RSAs in a straightforward way. However, we should be able to obtain a bijection between runs $(S_0, P_0) \rightarrow (S_1, P_1) \rightarrow (S_2, P_2) \rightarrow \ldots$ and $(S'_0, P'_0) \rightarrow (S'_1, P'_1) \rightarrow (S'_2, P'_2) \rightarrow \ldots$ for two RSAs $\mathcal{A}$ and $\mathcal{A}'$. Then we should clearly have that $S_i = S'_i$ holds for all i, and that P_i and P'_i are behaviourally equivalent as non-reflective, sequential algorithms. This is not yet satisfactory, as P_i and P'_i may still operate on different signatures.

We can argue that it is sufficient to consider the restrictions of P_i and P'_i on the "standard" part of the signatures, i.e. the functions that do not take terms as values. This would allow the algorithms P_i and P'_i to differ in their changes to themselves, but these differences have de facto no effect, as the updates yielded by these algorithms produce the same state transition and result in modified, yet behaviourally equivalent algorithms throughout the complete run. In other words, the possibly differing changes to the algorithm may extend the signature by functions or integrate fragments of "code" that are never used and thus have no effect on the updates.

Definition 3 (Behavioural Equivalent RSAs). *Let* $r_1 = (S_0, P_0), (S_1, P_1), (S_2, P_2), \ldots,$ *and* $r_2 = (S'_0, P'_0), (S'_1, P'_1), (S'_2, P'_2), \ldots,$ *be runs of RSAs. We consider that* r_1 *and* r_2 *are* essentially equivalent runs *if for every* $i \geq 0$ *the following holds:*

1. $S_i = S'_i$.
2. *The restrictions* $P_i|_\Sigma$ *and* $P'_i|_\Sigma$ *of, respectively,* P_i *and* P'_i *to the signature* Σ *of* S_i *and* S'_i*, constitute behavioural equivalent non-reflective sequential algorithms.*

Two RSAs $\mathcal{A}$ *and* $\mathcal{A}'$ *are* behavioural equivalent RSAs *iff* $\mathcal{A}$ *and* $\mathcal{A}'$ *have essentially equivalent classes of essentially equivalent runs. More precisely, iff there is a bijection* ζ *between runs of* $\mathcal{A}$ *and* $\mathcal{A}'$*, respectively, such that* r *and* $\zeta(r)$ *are essentially equivalent for all run* r*.*

3 Reflective Abstract State Machines

In this section we define a model of *reflective* ASMs (rASMs for short) and show that every rASM is a RSA in the precise sense of Definition 2. Given a signature Σ, i.e. a set of function symbols, then a sequential ASM-rule over Σ is defined as follows [5]:

assignments. $f(t_1, \ldots, t_{ar_f}) := t_0$ (with terms t_i built over Σ) is a rule.
branching. If r_+ and r_- are rules and φ is a Boolean term, then also **if** φ **then** r_+ **else** r_- **endif** is a rule.
bounded parallel composition. If $r_1, \ldots, r_n$ are rules, then also **par** $r_1 \ldots r_n$ **endpar** is a rule.

Each rule can be interpreted in a state, and doing so yields an update set. In general, a *location* is a pair $\ell = (f, (a_1, \ldots, a_k))$ with a function symbol $f \in \Sigma$ and a k-tuple (k being the arity of f) of values from the fixed base set B, and an *update* is a pair (ℓ, a_0) with a value $a_0 \in B$.

The rules of an rASM are also sequential ASM rules, and the interpretation of these rules in terms of update sets coincides with those of sequential ASMs as defined in [5]. The key difference is that rASMs work over extended-states, where each extended-state includes a finite representation of the rule that determines the update set produced by the machine in the current extended-state. In this way, we also allow an rASM to produce updates to its current rule.

Let (S, R) be an extended state of a rASM $\mathcal{M}$. We assume that the sub-structure S includes the following background of computation:

- An infinite reserve of values and function names.
- All ordered pairs of elements in the base set.
- The usual Boolean functions and usual constants `true`, `false` and `undef`.
- The "program" functions *update*, *par*, *if*.

The "program" functions are static and interpreted as follows:

- $update(f(t_1, \ldots, t_n), t_0) = (t_0, t_1, \ldots, t_n)$
- $par(t_1, t_2) = (val_S(t_1), val_S(t_2))$
- $if(t_1, t_2) = (t_1, val_S(t_2))$.

Notice that the following function induces a one-to-one correspondence between ASM rules and "program" terms, so that every ASM rule can be represented as a "program" term.

- $progToFunction(f(t_1, \ldots, t_n) := t_0) = update(f(t_1, \ldots, t_n), t_0)$.
- $progToFunction(\mathbf{if}\ \varphi\ \mathbf{then}\ R\ \mathbf{endif}) = if(\varphi, progToFunction(R))$.
- $progToFunction(\mathbf{par}\ R_1\ R_2\ \mathbf{endpar}) = par(progToFunction(R_1), progToFunction\ (R_2))$.

The sub-structure R of the extended-state (S, R) (i.e., the structure which contains the encoding of the "current" ASM rule) includes:

- The set of all ground terms.
- A distinguished location *self* interpreted as a "program" term (the current ASM rule).
- A finite alphabet A (the alphabet of the ground terms) and all strings in A^*.
- A constant s_i for each symbol $s_i \in A$ and a constant λ for the empty string.
- The usual string manipulation functions, including the concatenation function ".".
- A total injective function *TermToString* from the set of all terms of vocabulary Σ to A^*.
- A partial function *StringToTerm* defined as the inverse of *TermToString*.
- A function $argumentNo(t, n)$ which returns the n-th argument of the term t.
- A function $insertArgument(s, n, t)$ which returns a copy of t with its n-th argument replaced by s.

Since in each extended-state (S, R) of a rASM, the sub-structure R represents a uniquely determined sequential ASM rule, we usually refer to it as a rule rather than as a structure, meaning the rule corresponding to the "program term" in the location *self*.

Definition 4. *An rASM $\mathcal{M}$ is formed by:*

- *A non-empty set $\mathcal{R}_\mathcal{M}$ of* sequential ASM rules *(represented as first-order structures).*
- *An* initial rule $R_0 \in \mathcal{R}_\mathcal{M}$.
- *A non-empty set $\mathcal{S}_\mathcal{M}$ of* states *(i.e., first-order structures) closed under isomorphisms.*
- *A non-empty set $\mathcal{I}_\mathcal{M} \subseteq \mathcal{S}_\mathcal{M}$ of* initial states*, also closed under isomorphisms.*
- *A set of extended-states $\mathcal{E}_\mathcal{M} = \mathcal{S}_\mathcal{M} \times \mathcal{R}_\mathcal{M}$.*
- *A transition function $\tau_\mathcal{M}$ over $\mathcal{E}_\mathcal{M}$ such that $\tau_\mathcal{M}((S, R)) = (S, R) + \Delta(R, (S, R))$ for every $(S, R) \in \mathcal{E}_\mathcal{M}$, , where $R = val_S(self)$ is the closed ASM rule in location self in the extended state (S, R), $\Delta(R, (S, R))$ is the update set yielded by this rule in S, and $(S, R) + \Delta(R, (S, R))$ denotes the extended-state obtained by applying to (R, s) the update set $\Delta(R, (S, R))$.*

A *run* or *computation* of a reflective sequential ASM is a finite or infinite sequence of extended states $(S_0, R_0), (S_1, R_1), (S_2, R_2), \ldots$, where S_0 is a state in $\mathcal{I}_\mathcal{M}$, R_0 is the initial rule, and $(S_{i+1}, R_{i+1}) = \tau_\mathcal{M}((S_i, R_i))$ holds for every $i \geq 0$.

Notice that for every $R \in \mathcal{R}_\mathcal{M}$, the functions in R allow us to examine and modify the "program" term stored in *self*. For instance, assume that the current value stored in *self* is the term $update(f(t), s)$ and that we want to change it to $update(f(t), s+1)$. Assuming the alphabet A includes the symbols "+" and "1", the following sequential ASM rule updates *self* to the desired "program" term: $self := insertArgument(stringToTerm(TermToString(argumentNo(self, 2)) \cdot + \cdot 1), 2, self)$.

Of course, it is quite cumbersome to update the rule in self by using the small set of background functions provided here. Nevertheless, this is enough to show that our approach works. In practice, we can use more convenient representations, for instance by means of complex values such as syntax trees, as well as more sophisticated functions to inspect and modify the ASM rules. Note that the kind of reflection that the RRM uses is a bit different to the one we propose in this work. We could call it "partial reflection", since the sequence of actions performed in each transition, except for the queries to the relational store, never changes. We could then think of a different definition of the reflective ASM to represent partial reflection, where we only add to the sequential ASM a rule **eval** t, which takes a "program" term t as its argument, and interpret it as a sequential ASM rule (other than **eval**) which is then executed.

The next result shows the plausibility of our reflective ASM thesis.

Theorem 1. *Every reflective ASM $\mathcal{M}$ is a RSA.*

Proof (Sketch). We need to show that $\mathcal{M}$ satisfies the reflective sequential time, reflective abstract extended-state and reflective bounded exploration postulates. The first two postulates are already built into the definition of rASM, and the preservation of isomorphisms is straightforward.

In order to show that $\mathcal{M}$ satisfies also the reflective bounded exploration postulate, we let $W_{st} = \emptyset$ and $W_{wt} = \{self\}$. We see next that if two extended-states (S, R) and (S', R') of $\mathcal{M}$ strongly coincide over W_{wt} then $\Delta(R, (S, R)) = \Delta(R', (S', R'))$. Since the states strongly coincide over W_{wt} we have that:

1. $val_{(S,R)}(self) = val_{(S',R')}(self)$.
2. $val_{(S,R)}(raise_{(S,R)}(self)) = val_{(S',R')}(raise_{(S',R')}(self))$.

Let $W_r = \{r\}$ and $W_{r'} = \{r'\}$, where r and r' are the tuples of terms that result from the evaluation of *self* in (S, R) and (S', R'), respectively. From our definition of the "program" functions and the proof of the plausibility theorem of the sequential ASM thesis, we get that W_r and $W_{r'}$ constitute, respectively, bounded exploration witnesses for the sequential ASM rules R and R'. In turn, by (1), we further have that $W_r = W_{r'}$. Finally, by (2) we get that (S, R) and (S', R') coincide on $W_r = W_{r'}$. Hence, by Gurevich's bounded exploration postulate for sequential algorithms, we get that $\Delta(R, (S, R)) = \Delta(R', (S', R'))$. The plausibility theorem for RSA then follows. □

4 The Reflective Sequential ASM Thesis

We start by analysing an arbitrary RSA $\mathcal{A}$. Let $W_{st} \cup W_{wt}$ be a bounded exploration witness for $\mathcal{A}$ and let (S, P) be a state of $\mathcal{A}$. We define the set of *terms generated by W_{wt} in (S, P)* as follows: $G^{(S,P)}_{W_{wt}} = \{raise_{(S,P)}(t) \mid t \in W_{wt}\}$. We assume that $W_{st} \cup G^{(S,P)}_{W_{wt}}$ is closed under sub-terms and call it the set of *critical terms of (S, P)*.

The following lemma can be proven using the same argument as in the proof of the analogous Lemma 6.2 in the sequential ASM thesis [8].

Lemma 1. *If $(f, (v_1, \ldots v_n), v_0)$ is an update in $\Delta(\mathcal{A}, (S, P))$, then $v_0, v_1, \ldots, v_n$ are values of critical terms of (S, P).*

Lemma 1 implies that every update in $\Delta(\mathcal{A}, (S, P))$ can be programmed by an update rule of the form $f(t_1, \ldots t_n) := t_0$, where the terms $t_0, t_1, \ldots, t_n$ are critical terms of (S, P). To program the whole $\Delta(\mathcal{A}, (S, P))$, we define a sequential ASM rule $r_{(S,P)}$ which is the parallel combination (by means of **par** rules) of all update rules in the following *finite* set:

$$\{f(t_1, \ldots, t_n) := t_0 \mid t_0, t_1, \ldots, t_n \in W_{st} \cup G^{(S,P)}_{W_{wt}} \text{ and } (f, (val_{(S,P)}(t_1), \ldots, val_{(S,P)}(t_n)), val_{(S,P)}(t_0)) \in \Delta(\mathcal{A}, (S, P))\}.$$

As $W_{st} \cup G^{(S,P)}_{W_{wt}}$ is finite and the signature of (S, P) is also finite, $r_{(S,P)}$ is well defined.

Corollary 1. *For every $(S, P) \in \mathcal{S}_{\mathcal{A}}$ there is a rule $r_{(S,P)}$ such that:*

1. $r_{(S,P)}$ uses only critical terms, i.e., terms in $W_{st} \cup G^{(S,P)}_{W_{wt}}$.
2. $\Delta(r_{(S,P)}, (S, P)) = \Delta(\mathcal{A}, (S, P))$.

From now on, $r_{(S,P)}$ is as in the previous corollary.

Lemma 2. *If two extended-states (S, P) and (S', P') of $\mathcal{A}$ strongly coincide over $W_{st} \cup W_{wt}$, then $\Delta(r_{(S,P)}, (S', P')) = \Delta(\mathcal{A}, (S', P'))$.*

Proof. As (S, P) and (S', P') strongly coincide over $W_{st} \cup W_{wt}$, we have that $G^{(S,P)}_{W_{wt}} = G^{(S',P')}_{W_{wt}}$ and that, for every $t \in W_{st} \cup G^{(S,P)}_{W_{wt}}$, $val_{(S,P)}(t) = val_{(S',P')}(t)$. As $r_{(S,P)}$ only involves critical terms of (S, P), i.e., terms in $W_{st} \cup G^{(S,P)}_{W_{wt}}$, we have that $\Delta(r_{(S,P)}, (S, P)) = \Delta(r_{(S,P)}, (S', P'))$. By Corollary 1, $\Delta(r_{(S,P)}, (S, P)) = \Delta(\mathcal{A}, (S, P)$. Finally, we obtain $\Delta(\mathcal{A}, (S, P)) = \Delta(\mathcal{A}, (S', P'))$ by the reflective bounded exploration postulate. □

Let (S, P) and (S', P') be extended-states of $\mathcal{A}$. We say that (S', P') is *relative $W[(S, P)]$-equivalent* to (S, P) if $G^{(S',P')}_{W_{wt}} = G^{(S,P)}_{W_{wt}}$, and that they *coincide over $W[(S, P)]$* (in the sense of the sequential ASM thesis [8]) if $val_{(S,P)}(t) = val_{(S',P')}(t)$ for all $t \in W_{st} \cup G^{(S,P)}_{W_{wt}}$ (i.e., for all critical terms of (S, P)).

The following is a straightforward corollary of Lemma 2 obtained by restricting the sets of updates to the locations in the "standard" sub-structure of the extended-states. Δ_{st} denotes the subset of updates with function names which do *not* appear in Σ_{algo}.

Corollary 2. *If two extended-states (S, P) and (S', P') are relative $W[(S, P)]$-equivalent and coincide over $W[(S, P)]$, then we have $\Delta_{st}(r_{(S,P)}, (S', P')) = \Delta_{st}(\mathcal{A}, (S', P'))$.*

Consider the class $\mathcal{C}[(S, P)]$ of relative $W[(S, P)]$-equivalent states of $\mathcal{A}$. Two states (S_1, P_1) and (S_2, P_2) of $\mathcal{A}$ are *W-equivalent relative to $\mathcal{C}[(S, P)]$* iff $(S_1, P_1), (S_2, P_2) \in \mathcal{C}[(S, P)]$ and $E_{(S_1,P_1)} = E_{(S_2,P_2)}$, where (for $i = 1, 2$) $E_{(S_i,P_i)}(t_1, t_2) \equiv val_{(S_i,P_i)}(t_1) = val_{(S_i,P_i)}(t_2)$ is an equivalence relation in the set of critical terms of (S, P).

Lemma 3. *If two extended-states (S_1, P_1) and (S_2, P_2) of $\mathcal{A}$ are W-equivalent relative to $\mathcal{C}[(S, P)]$, then $\Delta_{st}(r_{(S_1,P_1)}, (S_2, P_2)) = \Delta_{st}(\mathcal{A}, (S_2, P_2))$.*

Proof (sketch). Note that if we assume $\Delta_{st}(r_{(S_1,P_1)}, (S_3, P_3)) = \Delta_{st}(\mathcal{A}, (S_3, P_3))$ for a state $(S_3, A_3) \in \mathcal{C}[(S, P)]$ with S_3 isomorphic to S_2, then we get that $\Delta_{st}(r_{(S_1,P_1)}, (S_2, P_2)) = \Delta_{st}(\mathcal{A}, (S_2, P_2))$. This fact is analogous to Lemma 6.8 of the sequential ASM thesis [8] and can be proven in the same way. Thus, we just need to find an extended-state $(S_3, P_3) \in \mathcal{C}[(S, P)]$ with S_3 isomorphic to S_2 and such that $\Delta_{st}(r_{(S_1,P_1)}, (S_3, P_3)) = \Delta_{st}(\mathcal{A}, (S_3, P_3))$.

Assume w.l.o.g. that the base sets of S_1 and S_2 are disjoint. Let S_3 be the structure isomorphic to S_2 which is obtained by replacing $val_{S_2}(t)$ with $val_{S_1}(t)$

for all critical terms t of (S, P). This is well defined because (S_1, P_1) and (S_2, P_2) are W-equivalent relative to $\mathcal{C}[(S, P)]$. Take $P_3 = P_2$, then $(S_3, P_3) \in \mathcal{C}[(S, P)]$. By the reflective abstract state postulate, (S_3, P_3) is an extended-state of $\mathcal{A}$. Since (S_1, P_1) and (S_3, P_3) coincide over the set of critical terms of (S, P), Corollary 2 gives $\Delta_{st}(r_{(S_1,P_1)}, (S_3, P_3)) = \Delta_{st}(\mathcal{A}, (S_3, P_3))$. □

Let $\varphi_{(S,P)}$ be the following Boolean term:

$$\bigwedge_{\substack{t_i,t_j \in W_{st} \cup G^{(S,P)}_{W_{wt}} \\ val_{(S,P)}(t_i) = val_{(S,P)}(t_j)}} t_i = t_j \quad \wedge \quad \bigwedge_{\substack{t_i,t_j \in W_{st} \cup G^{(S,P)}_{W_{wt}} \\ val_{(S,P)}(t_i) \neq val_{(S,P)}(t_j)}} \neg(t_i = t_j).$$

As the set of critical terms of an extended-state (S, P) (i.e., $W_{st} \cup G^{(S,P)}_{W_{wt}}$) is finite, there is a finite set $\{(S_1, P_1), \ldots, (S_n, P_n)\}$ of states in $\mathcal{C}[(S, P)]$ (the class of relative $W[(S, P)]$-equivalent states of $\mathcal{A}$) such that every state in $\mathcal{C}[(S, P)]$ is W-equivalent relative to $\mathcal{C}[(S, P)]$ to one of the states (S_i, P_i). Construct a rule **par if** $\varphi_{(S_1,P_1)}$ **then** $r_{(S_1,P_1)}$ **endif** ... **if** $\varphi_{(S_n,P_n)}$ **then** $r_{(S_n,P_n)}$ **endif endpar** Then the following result clearly follows from the previous lemmata.

Lemma 4. *$\Delta_{st}(r_{[(S,P)]}, (S_i, P_i)) = \Delta_{st}(\mathcal{A}, (S_i, P_i))$ for every extended-state $(S_i, P_i) \in \mathcal{C}[(S, P)]$, i.e., for every extended-state that is relative $W[(S, P)]$-equivalent to (S, P).*

Thus, for every class $\mathcal{C}([S_i, P_i])$ of extended-states of $\mathcal{A}$, we have a corresponding rule $r_{[(S_i,P_i)]}$ such that Lemma 4 holds. Now, we need to extend this result to all extended-states which belong to some run of $\mathcal{A}$, not just for the extended-states in the class $\mathcal{C}([S_i, P_i])$. Here is when the power of reflection becomes apparent.

Fix an arbitrary initial extended state (S, P) of $\mathcal{A}$. We define $\mathcal{M}$ as the *reflective* ASM machine with $\mathcal{E}_{\mathcal{M}} = \{(S_i, P_i') \mid (S_i, P_i) \in \mathcal{E}_{\mathcal{A}}$ and P_i' is the "self" representation of $r_{[(S_i,P_i)]}\}$ and $\mathcal{I}_{\mathcal{M}} = \{(S_i, P') \mid S_i \in \mathcal{I}_{\mathcal{A}}$ and P' is the "self" representation of $r_{[(S,P)]}\}$.

Lemma 5. *For every run of $\mathcal{A}$ of the form $(S_0, P_0), (S_1, P_1), \ldots$ and corresponding run of $\mathcal{M}$ of the form $(S_0', P_0'), (S_1', P_1'), \ldots$ with $S_0 = S_0'$, it holds that $\Delta_{st}(r_{[(S_i,P_i')]}, (S_i', P_i')) = \Delta_{st}(\mathcal{A}, (S_i, P_i))$.*

Proof (Sketch). We prove it by induction on an arbitrary run of $\mathcal{A}$. By the reflective sequential time postulate, we know that every initial extended-state (S_0, P_0) of every run of $\mathcal{A}$ is relative $W[(S, P)]$-equivalent to the initial extended-state (S, P) used in the construction of $\mathcal{M}$. Thus, we get from Lemma 4 that $\Delta_{st}(r_{[(S_0,P_0')]}, (S_0, P_0)) = \Delta_{st}(\mathcal{A}, (S_0, P_0))$. Given the restriction to "standard" updates which do not involve updates to the algorithm, we have

$$\Delta_{st}(r_{[(S_0,P_0')]}, (S_0', P_0')) = \Delta_{st}(\mathcal{A}, (S_0, P_0)).$$

Regarding the inductive step. As P_i is a sequential algorithm, it is captured by a sequential ASM M_i. Moreover, due to Gurevich's proof of the sequential ASM thesis [8], the rule has the form

$$\textbf{par if } \psi_1 \textbf{ then } r_1 \textbf{ endif } \ldots \textbf{ if } \psi_k \textbf{ then } r_k \textbf{ endif endpar},$$

where each r_j is a **par** block of assignment rules. All ψ_j and r_j involve critical terms defined by a bounded exploration witness of P_i such as $W_{st} \cup G^{(S_i,P_i)}_{W_{wt}}$.

Due to construction of $r_{[(S'_i,P'_i)]}$, we have that $W_{st} \cup G^{(S_i,P_i)}_{W_{wt}}$ is bounded exploration witness of the "self" representation P'_i of $r_{[(S'_i,P'_i)]}$. In turn, by construction of $\mathcal{M}$ it can be shown that $W_{st} \cup W_{wt}$ is a bounded exploration witness for $\mathcal{M}$. Thus, the updates in $\Delta_{st}(r_{[(S'_i,P'_i)]}, (S'_i, P'_i))$, transform the "self" representation P'_i of $r_{[(S'_i,P'_i)]}$ into the "self" representation P'_{i+1} of $r_{[(S'_i,P'_{i+1})]}$. Since from the inductive hypothesis it can be shown that $S'_i = S_i$, we get that $\Delta_{st}(r_{[(S_{i+1},P'_{i+1})]}, (S'_{i+1}, P'_{i+1})) = \Delta_{st}(\mathcal{A}, (S_{i+1}, P_{i+1}))$. □

Using the previous key lemma, it is not difficult to show that every run of $\mathcal{A}$ of the form $(S_0, P_0), (S_1, P_1), \ldots$ is *essentially equivalent* to the corresponding run of $\mathcal{M}$ of the form $(S'_0, P'_0), (S'_1, P'_1), \ldots$ with $S_0 = S'_0$, i.e., that $S_i = S'_i$ and that the restriction of P_i and P'_i to the signature Σ of S_i and S'_i results in non-reflective algorithms which are behavioural equivalent. This implies our main result.

Theorem 2. *For every RSA $\mathcal{A}$ there is a behavioural equivalent rASM machine $\mathcal{M}$.*

5 Conclusion

In this paper we investigated a behavioural theory for reflective sequential algorithms (RSAs) following our conjecture in [7]. Grounded in related work concerning behavioural theories for sequential algorithms [8], (synchronous) parallel algorithms [6], non-deterministic algorithms [10] and concurrent algorithms [4] we developed a set of abstract postulates characterising RSAs, extended ASMs to reflective Abstract State Machines (rASMs), and formally sketched the proof that any RSA as stipulated by the postulates can be step-by-step simulated by a rASM. The key contributions are the postulates themselves, as they provide a language-independent definition of RSAs and the characterisation proof.

With this behavioural theory we lay the foundations for rigorous development of reflective algorithms and thus self-adaptive systems. However, several open tasks still have to be addressed before a general behavioural theory of *evolving concurrent systems* (ECS) will be reached. It is required to combine the behavioural theory developed in this paper with those for parallel algorithms thus proving a behavioural theory for reflective parallel algorithms, and with the theory of concurrency thus proving a behavioural theory for concurrent reflective systems, i.e. ECS. In view of the similarity of arguments in the separate behavioural theses this integration appears plausible, but nonetheless constitutes

a mathematically challenging problem. Furthermore, for rigorous development extensions to the refinement method for ASMs [3] and to the logic used for verification [14] will be necessary. These will be addressed in follow-on research.

References

1. Blass, A., Gurevich, Y.: Abstract state machines capture parallel algorithms. ACM Trans. Comput. Logic **4**(4), 578–651 (2003)
2. Blass, A., Gurevich, Y.: Abstract state machines capture parallel algorithms: correction and extension. ACM Trans. Comp. Logic **9**(3), 19 (2008)
3. Börger, E.: The ASM refinement method. Formal Aspects Comput. **15**, 237–257 (2003)
4. Börger, E., Schewe, K.D.: Concurrent abstract state machines. Acta Inform. **53**(5), 469–492 (2016)
5. Börger, E., Stärk, R.: Abstract State Machines. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-642-18216-7
6. Ferrarotti, F., Schewe, K.D., Tec, L., Wang, Q.: A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. Theor. Comput. Sci. **649**, 25–53 (2016)
7. Ferrarotti, F., Tec, L., Torres, J.M.T.: Towards an ASM thesis for reflective sequential algorithms. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 244–249. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_16
8. Gurevich, Y.: Sequential abstract-state machines capture sequential algorithms. ACM Trans. Comput. Logic **1**(1), 77–111 (2000)
9. Riccobene, E., Scandurra, P.: Towards ASM-based formal specification of self-adaptive systems. In: Ameur, Y.A., Schewe, K.D. (eds.) Abstract State Machines, Alloy, B, TLA, VDM, and Z. LNCS, vol. 8477, pp. 204–209. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43652-3
10. Schewe, K.D., Wang, Q.: A customised ASM thesis for database transformations. Acta Cybern. **19**(4), 765–805 (2010)
11. Smith, B.C.: Reflection and semantics in LISP. In: Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL 1984, pp. 23–35. ACM (1984)
12. Stemple, D., et al.: Type-safe linguistic reflection: a generator technology. In: Atkinson, M., Welland, R. (eds.) Fully Integrated Data Environments. Esprit Basic Research Series, pp. 158–188. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-642-59623-0_8
13. Van den Bussche, J., Van Gucht, D., Vossen, G.: Reflective programming in the relational algebra. J. Comput. Syst. Sci. **52**(3), 537–549 (1996)
14. Wang, Q., Ferrarotti, F., Schewe, K.D., Tec, L.: A complete logic for non-deterministic database transformations. CoRR abs/1602.07486 (2016). http://arxiv.org/abs/1602.07486