Evolving Concurrent Systems – Behavioural Theory and Logic*

Klaus-Dieter Schewe
Software Competence Center
Hagenberg, Softwarepark 21,
4232 Hagenberg, Austria
kd.schewe@scch.at

Flavio Ferrarotti
Software Competence Center
Hagenberg, Softwarepark 21,
4232 Hagenberg, Austria
flavio.ferrarotti@scch.at

Loredana Tec
Software Competence Center
Hagenberg, Softwarepark 21,
4232 Hagenberg, Austria
loredana.tec@scch.at

Qing Wang
Research School of Computer
Science, The National
University of Australia,
Canberra, ACT, Australia
qing.wang@anu.edu.au

Wenya An
Software Competence Center
Hagenberg, Softwarepark 21,
4232 Hagenberg, Austria
wenya.an@scch.at

ABSTRACT

A *concurrent system* can be characterised by autonomously acting agents, where each agent executes its own program, uses shared resources and communicates with the others, but otherwise is totally oblivious to the behaviour of the other agents. In an *evolving* concurrent system agents may change their programs, enter or leave the collection at any time thereby changing the behaviour of the overall system. In this paper we present a behavioural theory of evolving concurrent systems, i.e. we provide (1) a small set of postulates that characterise evolving concurrent systems in a precise conceptual way without any reference to a particular language, (2) an abstract machine model together with a plausibility proof that the abstract machines satisfy the postulates, and (3) a characterisation proof that any system stipulated by the postulates can be step-by-step simulated by an abstract machine. The theory integrates the behavioural theories for unbounded (synchronous) parallel algorithms, asynchronous concurrent systems, and reflective algorithms, respectively. However, in the latter two theories only sequential agents and sequential reflective algorithms were considered. Furthermore, linguistic reflection has not been integrated with parallelism. We will show how these research gaps can be closed.

*The research reported in this paper results from the project *Behavioural Theory and Logics for Distributed Adaptive Systems* supported by the **Austrian Science Fund (FWF: [P26452-N15])**. The research has further been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH **(FFG: [844597])**.

ACSW'17 January 31–February 03, 2017, Geelong, Australia

ISBN 978-1-4503-4768-6/17/01…$15.00

DOI: http://dx.doi.org/10.1145/3014812.3017446

The behavioural theory implies that *concurrent reflective Abstract State Machines* (crASMs) can be used as a specification and development language for evolving concurrent systems. We therefore investigate a logic for crASMs. Based on the simple observation that concurrent ASMs can be mimicked by non-deterministic parallel ASMs we exploit the complete one-step logic for non-deterministic ASMs for the definition of a logic capturing concurrency. By making the extra-logical rules in the logic subject to being interpreted in a state we extend the logic to capture also reflection.

1. INTRODUCTION

A *concurrent system* can be characterised by autonomous agents, where each agent executes its own program, uses shared resources and communicates with the other agents, but otherwise is totally oblivious to the behaviour of others. There are numerous models of concurrency in the literature or implemented in current hardware/software systems and underlying distributed algorithms, specification and programming languages: distributed algorithms [23], process algebras [19, 32, 27, 28], actor models [18], trace theory [24, 25, 43], Petri nets [30, 29, 40], etc.

Abstract State Machines (ASMs) have been used since their introduction to model sequential and concurrent systems (see [8, Ch.6,9] for references), but only recently the theory counterpart of the celebrated sequential ASM thesis [16] has been discovered [7]. This *concurrent ASM thesis* formulates a postulate characterising concurrency based on the intuitive understanding of computations of multiple autonomous agents, which execute each a sequential process, run asynchronously, each with its own clock, and interact with (and know of) each other only via reading/writing values of designated locations. It has been proven that concurrent algorithms are captured by *concurrent Abstract State Machines*, i.e. families of agents each equipped with a sequential ASM, the semantics of which is defined by *concurrent ASM runs*, which overcome the problems of Gurevich's distributed ASM runs [15] and generalize Lamport's sequentially consistent runs [21]. This consitutes a behavioural theory that provides a foundation for concurrent sequential algorithms and their rigorous specification, refinement and

verification.

However, the theory in [7] still restricts the agents to parallelism that is a priori bounded by the specification. It is known that unbounded parallelism is captured by (synchronous) parallel ASMs [4, 5, 12], and it has been conjectured that the restriction to sequential agents can thus be easily dropped in the concurrent ASM thesis. In this paper we formally integrate the (simplified) behavioural theory of parallel algorithms [12] into the concurrent ASM thesis, which will give us a foundation for concurrent algorithms in general and enable rigorous specification, refinement and verification for this extended class of concurrent systems. Examples for using concurrent (parallel) ASMs for the specification and generalisation of distributed algorithms [23] have been studied in [3].

Recently, (self-)adaptive systems have attracted a lot of interest in research, in particular in connection with systems of (cyber-physical) systems [31], biologically-inspired systems [39] or observer/controller architectures [36, 38] (aka monitoring and adaption). Adaptivity refers to the ability of a system to change its own behaviour. As state-based rigorous methods for software systems development such as B [1], Event-B [2], Abstract State Machines (ASMs) [8], TLA$^+$ [22] and others are coupled with a genericity promise, i.e. they can be applied universally to a large class of systems, it appears to be desirable to further extend the theory to capture *evolving concurrent systems*, i.e. to permit the agents in a concurrent system to be also adaptive. In [13] a behavioural theory for reflective, sequential algorithms was conjectured, which could be proven in [9], i.e. reflective, sequential algrithms can be captured by sequential reflective ASMs, which covers adaptivity in the most general sense. Thus, in order to obtain a general foundation for evolving concurrent systems we further integrate the theories of parallel algorithms and reflective algorithms, and then extend the theory to capture also reflection.

Besides rigorous design and stepwise refinement-based development verification is a key concern for rigorous systems development. All rigorous methods are supported by appropriate logics such as the logic for Event-B [35], the logic for ASMs [37], and the logic for TLA$^+$ [26]. These logics support primarily the verification of properties for a single machine step exploiting also concepts from dynamic logic [17], though extensions in temporal logic (see e.g. [33] for ASMs and [20] for TLA$^+$) have also been investigated to enable the reasoning about complete runs. In order to reason about evolving concurrent systems we investigate an extension of the one-step logic for ASMs to capture concurrency and reflection. This logic has been extended in [41] to deal with non-deterministic database transformations, for which the unsolved problem of non-determinism and the handling of multi-set functions for synchronisation had to be solved. This was further streamlined, partly corrected and extended in [42] and taken out of the database context in [10]. It can then be easily observed that concurrent ASMs could be mimicked by non-deterministic ASMs working on multiple local copies of the states, thus the logic can be exploited to capture concurrency in general. However, the obtain full reasoning power, we will show how to move from a one-step logic to a multiple-step logic, as a single step of an agent in a concurrent system may correspond to multiple steps of the whole concurrent system. Furthermore, we will substitute the extra-logical rules that are used in the ASM logic by variables that are to be interpreted in a state, but yield rules, by means of which we can capture reflection.

In Section 2 we will investigate the behavioural theory of general reflective algorithms. First this requires an integration of the set of postulates for parallel and reflective algorithms with a particular emphasis on bounded exploration. This will be followed by a brief description, why parallel, reflective ASMs satisfy the postulates, and finally by a sketch of the proof that any parallel, reflective algorithm as stipulated by the postulates will be captured by a behaviourally equivalent reflective ASM. In Section 3 we will then investigate evolving concurrent systems by adding the concurrency postulate. This will again be followed by a brief outline of concurrent reflective ASMs and a sketch of the proof that these capture evolving concurrent systems. Section 4 is then dedicated to the development of a logic for concurrent reflective ASMs, where we first consider concurrency and reflection in isolation, and then integrate the necessary extensions to the logic. We conclude with a brief summary and outlook in Section 5.

2. REFLECTIVE PARALLEL ALGORITHMS

The celebrated sequential ASM thesis needs only three simple, intuitive postulates to characterise sequential algorithms:

Sequential time: Each sequential computation proceeds by means of a *transition function* $\mathcal{S} \to \mathcal{S}$, 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.

Actually, bounded exploration is grounded in the simple observation that whatever is needed to determine the update set in some state must be contained in the finite representation of the algorithm. In a nutshell, we can always imagine W to be the set of terms that are "read" by the sequential algorithm in a state in order to determine the updates.

Note that the sequential time and abstract state postulates are also used in the parallel ASM thesis, whereas the main difference lies in bounded exploration and a necessary background postulate.

2.1 Sequential Time

Clearly, when extending the notion of sequential or parallel 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 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. Then a run of a reflective algorithm corresponds to the sequence of pairs (S_i, P_i), where in each step both the state S_i and the algorithm P_i are updated. As observed in [9] 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 the signature and some syntactic description. For the former 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.

Thus, our first postulate states that every reflective algorithm works in sequential time, the key difference being that they work over *extended states* $\hat{S}_i$ that correspond to pairs (S_i, P_i). Transitions from one extended state to the next can involve both: updates to the state and updates to the algorithm. Let P be a sequential algorithm of some signature Σ and let $\Sigma' \subseteq \Sigma$. We use $P|_{\Sigma'}$ to denote the restriction of P to the (sub)signature Σ'.

Postulate 1. (Reflective Sequential Time Postulate). *A reflective algorithm $\mathcal{A}$ consists of the following:*

- *A non-empty set $\mathcal{S}_\mathcal{A}$ of* extended states*.*
- *A non-empty subset $\mathcal{I}_\mathcal{A} \subseteq \mathcal{S}_\mathcal{A}$ of* initial extended states *such that for all $(S, P), (S', P') \in \mathcal{I}_\mathcal{A}$, it holds that S and S' are first-order structures of a same signature Σ and $P|_\Sigma$ and $P'|_\Sigma$ have exactly the same runs.*
- *A one-step transformation function $\tau_\mathcal{A} : \mathcal{S}_\mathcal{A} \to \mathcal{S}_\mathcal{A}$ such that $\tau_\mathcal{A}((S, P)) = (S', P')$ iff $\tau_P((S, P)) = (S', P')$ for the one-step transformation function τ_P of the sequential algorithm P.*

The concept of run remains the same as in the ASM thesis for sequential algorithms, except that in the case of reflection we consider extended states instead of arbitrary states. That is, a *run* or *computation* of $\mathcal{A}$ is a sequence of states (S_0, P_0), (S_1, P_1), (S_2, P_2), ..., where (S_0, P_0) is an initial state in $\mathcal{I}_\mathcal{A}$ and $(S_{i+1}, P_{i+1}) = \tau_\mathcal{A}((S_i, P_i))$ holds for every $i \geq 0$.

On these grounds the concept of behavioural equivalence used for sequential algorithms cannot be generalised in a straightforward way. While behavioural equivalent sequential algorithms have exactly the same runs, this is not necessarily the case for reflection. 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 reflective algorithms. 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. S_i and S'_i are first-order structures of a same signature Σ_i and $P_i|_{\Sigma_i}$ and $P'_i|_{\Sigma_i}$ have exactly the same runs.

Definition 1 (Behavioural Equivalence). *Two reflective algorithms $\mathcal{A}$ and $\mathcal{A}'$ are* ***behaviourally equivalent*** *iff $\mathcal{A}$ and $\mathcal{A}'$ have essentially equivalent classes of essentially equivalent runs, i.e. 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.*

2.2 Abstract States

As in the sequential ASM thesis, states are first-order structures. However, they are not arbitrary first-order structures, since each state must also include (an encoding of) a sequential algorithm given by a finite text. Also similar to the sequential ASM thesis, we need the sequential algorithms encoded in the states to work at a fixed level of abstraction. This cannot be achieved by simply requiring the set of states and the one step transformation function to be closed under isomorphisms, since that would interfere with our concept of behavioural equivalence.

We say that two states (S, P) and (S', P') are *essentially isomorphic* if S and S' are isomorphic first-order structures of some vocabulary Σ and $P|_\Sigma$ and $P'|_\Sigma$ have exactly the same runs. If ζ is an isomorphism form S to S', then we say that (S, P) and (S', P') are *essentially isomorphic via ζ*.

Postulate 2. (Reflective Abstract State Postulate). *Let $\mathcal{A}$ be a reflective algorithm. Fix a signature Σ and an extension Σ_{ext} of the signature Σ with additional function names.*

- *States of $\mathcal{A}$ are first-order structures of signature Σ_{ext}.*
- *Every state (S, P) of $\mathcal{A}$ is formed by the* disjoint *union of an arbitrary first-order structure S of some* finite *signature $\Sigma_{st} \subseteq \Sigma$ and a first-order structure S_P of signature $\Sigma_{wt} = \Sigma_{ext} \setminus \Sigma$ which contains an encoding of the sequential algorithm P.*
- *The one-step transformation $\tau_\mathcal{A}$ of a RSA $\mathcal{A}$ does not change the base set of any state of $\mathcal{A}$.*
- *The sets $\mathcal{S}_\mathcal{A}$ and $\mathcal{I}_\mathcal{A}$ of, respectively, states and initial states of $\mathcal{A}$, are closed under essentially isomorphic states.*
- *If two states (S, P) and (S', P') of $\mathcal{A}$ are essentially isomorphic via an isomorphism ζ from S to S', then $\tau_\mathcal{A}((\mathbf{S}_1, A_1))$ and $\tau_\mathcal{A}((\mathbf{S}_2, A_2))$ are also essentially isomorphic via ζ.*

2.3 Background

Each computation uses some background [12]. When dealing with sequential algorithms the background is usually left implicit, as it contains the *reserve of values* not used in a current state, but available to be added to the active domain in any state transition, truth values and their connectives, and the value `undef`. When dealing with parallel algorithms we have to add at least a pairing constructor and a multiset constructor together with necessary operators on tuples and multisets. Also reflection requires some background, as it must be possible to refer to tuples to capture terms that represent a specification, e.g. ASM signatures and rules. It must further comprise extra-logical constant such as keywords that are used in the specification.

Most important, reflection requires the presence of a *raise* function that takes values in the extended domain that are terms and removes all extra-logical elements, so that *raise*(t) can be interpreted as a "normal term" yielding a value in the domain. For brevity we dispense with a more detailed discussion of the background (for details see [12, Sect. 3]).

Postulate 3. (Reflective Background Postulate). *Let A be a reflective algorithm of* vocabulary Σ_{ext} *with background class $\mathcal{K}$. The vocabulary $\Sigma_\mathcal{K}$ of $\mathcal{K}$ includes (at least)*

a binary tuple constructor *and a* multiset constructor *of unbounded arity; and the vocabulary* $\Sigma_{\mathbf{B}}$ *of the background of the computation states of A includes (at least) the following* obligatory *function symbols:*

- *Nullary function symbols* `true`, `false`, `undef` *and* $\oslash$.
- *Unary function symbols* `reserve`, `atomic`, `Boole`, $\neg$, `first`, `second`, $\{\!\{\cdot\}\!\}$, $\biguplus$ *and* `AsSet`.
- *Binary function symbols* $=$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\uplus$ *and* $(\,,)$.
- *raise mapping terms over* Σ_{ext} *to terms over* Σ.

All function symbols in $\Sigma_{\mathbf{B}}$*, with the sole exception of* `reserve`*, are static.*

2.4 Bounded Exploration

Regarding bounded exploration the problem is that in general we must expect that each algorithm P_i represented in state (S_i, P_i) has its own bounded exploration witness W_i. For parallel algorithms P_i such a bounded exploration witness is a set of multiset comprehension terms, where each element in such a multiset corresponds to a branch (or proclet) of the parallel computation. However, due to the construction of W_i in [12] we know that W_i is somehow contained in the finite representation of P_i. For instance, the ASM rule constructed in the proof of the parallel ASM thesis only contains terms derived from the 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 algorithm. So there must exist a finite set of terms W, which we will continue to call *bounded exploration witness*, such that its interpretation in an extended state yields both values and terms, and the latter represent W_i. Consequently, the interpretation of W and of its interpretation in an extended state suffice to determine the update set in that state. The main difference to the thesis for reflective sequential algorithms is that the terms may be multiset comprehension terms corresponding to extra-logical constructs for unbounded parallelism. This will lead to our *bounded exploration postulate* for reflective algorithms. Thus, we first need an extension of the notion of *strong coincidence* over a set of multiset comprehension terms.

Definition 2. (Strong Coincidence). *Let* (S, P) *and* (S', P') *be states of signature* Σ_{ext}*. Let* W_{st} *be a set of multiset comprehension terms over signature* Σ *and* W_{wt} *be a set of multiset comprehension terms over signature* $\Sigma_{ext} \setminus \Sigma$*.* (S, P) *and* (S', P') strongly coincide *over* $W_{st} \cup W_{wt}$ *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. $val_{(S,P)}(raise(t)) = val_{(S',P')}(raise(t))$*, where* $raise(t)$ *denotes the interpretation of* t *as a term of signature* Σ.

As usual, we use $\Delta(P, S)$ to denote the unique set of updates produced by the sequential algorithm P in state S. The unique set of updates produced by a RSA $\mathcal{A}$ in a state (S, P) is defined as $\Delta(\mathcal{A}, (S, P)) = \Delta(P, (S, P))$.

We can now formulate our fourth and last postulate for reflective algorithms. It generalises the bounded exploration postulate for parallel algorithms in [12] to reflective algorithms. The central difference with the analogous postulate in the parallel ASM thesis is the use of the stronger notion of coincidence. The idea is that, for every state (S_i, P_i), the multiset comprehension terms obtained by the interpretation in (S_i, P_i) of the terms in W_{wt} together with the "standard" terms in W_{st} form a bounded exploration witness for the sequential algorithm P_i.

Postulate 4. (Reflective Bounded Exploration Postulate). *For every reflective* $\mathcal{A}$ *of signature* Σ_{ext} *there is a finite set* W_{st} *of multiset comprehension terms over signature* Σ *and a finite set* W_{wt} *of multiset comprehension terms over signature* $\Sigma_{ext} \setminus \Sigma$ *such that* $\Delta(\mathcal{A}, (S, P)) = \Delta(\mathcal{A}, (S', P'))$ *holds, whenever states* (S, P) *and* (S', P') *of* $\mathcal{A}$ *strongly coincide on* $W_{st} \cup W_{wt}$.

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

Definition 3. *A* reflective algorithm *(RA) is characterised by the Reflective Sequential Time, Reflective Abstract State, Reflective Background and Reflective Bounded Exploration Postulates.*

2.5 Reflective Parallel ASMs

Let us now define a model of *reflective* ASMs (rASMs) for short) and show that every rASM is a RA in the precise sense of Definition 3. The set of ASM rules of the rASM, as well as the interpretation of these rules in terms of update sets, coincide with those of the parallel ASMs as defined in [8].

We assume that the signature Σ_{ext} of an rASM always includes a sub-signature Σ with an *infinite* reserve of function names of arity r for each $r \geq 0$. Let $\mathcal{M}$ be an rASM of signature Σ_{ext}, the sates of $\mathcal{M}$ are extended states (S_i, r_i) formed by an arbitrary first-order structure S_i of some finite signature $\Sigma_{st} \subseteq \Sigma$ and a first-order structure S_{r_i} of (also finite) signature $\Sigma_{wt} = \Sigma_{ext} \setminus \Sigma$ which contains an encoding of the sequential ASM rule r_i as a "program" term.

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

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

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

- $update(f(t_1, \ldots, t_n), t_0) = (t_0, t_1, \ldots, t_n)$
- $forall(t_1, t_2) = \{\!\{val_S(t_2) \mid val_S(t_1)\}\!\}$
- $if(t_1, t_2) = \{\!\{(t_1, t) \mid t \in val_S(t_2)\}\!\}$

Note that each ASM rule can be represented by a "program" term. For instance, if the ASM rule r has the form **if** φ **then** $f(t_1, \ldots, t_n) := t_0$ **endif** then the "program" term of the form $if(\varphi, update(f(t_1, \ldots, t_n), t_0))$ represents r.

The sub-structure S_r of (S, r) which contains the encoding of the parallel ASM r includes:

- The set of all ground terms of vocabulary Σ.
- A distinguished location *self* which stores a "program" term which represents the ASM rule r.
- A finite alphabet A and the set A^* of all strings over A.
- 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 constant s_i for each symbol $s_i \in A$ and a constant λ for the empty string.
- A string concatenation function "$\cdot$".
- A function *argumentNo* such that $argumentNo(t, n)$ is the n-th argument of the term t or `undef` if t does not have n-th argument or is not a term.
- A function *insertArgument*, which assigns to (s, n, t) the term t with its n-th argument replaced by s.

Notice that the functions available in S_r permit 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. An rASM $\mathcal{M}$ is then formed by:

- A non-empty set $\mathcal{S}_\mathcal{M}$ of extended states, closed under *essential* isomorphisms.
- A non-empty subset $\mathcal{I}_\mathcal{M} \subseteq \mathcal{S}_\mathcal{M}$ of *initial states* such that for all $(S, r), (S', r') \in \mathcal{I}_\mathcal{M}$, it holds that S and S' are first-order structures of a same signature Σ_{st} and $r|_{\Sigma_{st}}$ and $r'|_{\Sigma_{st}}$ have exactly the same runs.
- A transition function $\tau_\mathcal{M}$ over $\mathcal{S}_\mathcal{M}$ such that $\tau_\mathcal{M}((S, r)) = (S, r) + \Delta(r, (S, r))$ for every $(S, r) \in \mathcal{S}_\mathcal{M}$.

A *run* of a reflective sequential ASM is a finite or infinite sequence of extended states $(S_0, r_0), (S_1, r_1), \ldots$, where (S_0, r_0) is an state in $\mathcal{I}_\mathcal{M}$ and $(S_{i+1}, r_{i+1}) = \tau_\mathcal{M}((S_i, r_i))$ holds for every $i \geq 0$.

THEOREM 1. *Every reflective ASM $\mathcal{M}$ is a RA.*

The proof is analogous to the plausibility proof in [9]. The major difficulty is the construction of the bounded exploration witness, for which $\{self\}$ will be sufficient. However, it is crucial that the *raise* function turns the representation of `forall`-rules into multiset comprehension terms, which is captured by the "program functions".

2.6 The Reflective Parallel ASM Thesis

Our key result in this section is the characterisation theorem constituting the converse of Theorem 1. In order to prove it we can follow the argumentation in the corresponding proof in [9] taking in also the very sophisticated arguments in [12]. In a nutshell, we start with the update set $\Delta(\mathcal{A}, (S, P))$ in an arbitrary extended state (S, P). It is rather straightforward (Lemma 1) to show that every value occurring in an update is *critical*, i.e. results from the interpretation of the bounded exploration witness terms. The main difference here is that according to [12] we have to adopt the definition of critical value of being the elements of the resulting multisets, not the multisets as such.

In the case of reflective sequential algorithms a straightforward corollary gives us the existence of an ASM rule that yields the update set $\Delta(\mathcal{A}, (S, P))$ in the extended state (S, P). This is no longer the case for RAs. We have to go through the tedious construction in [12] that will show that any tuple with the same logical *type* as the tuple defined by an update in $\Delta(\mathcal{A}, (S, P))$ also gives rise to an update, from which we can conclude again the existence of an ASM rule (relying heavily on the `forall`-construct) that yields the update set at hand.

From this result the extensions required to obtain a behaviourally equivalent reflective ASM are largely the same as for the sequential case, i.e. the proof follows again the one in [9]. In the following we give a brief sketch of the proof without being able to go into details.

Let $\mathcal{A}$ be a RA of signature $\Sigma_{ext} = \Sigma \cup \Sigma_{wt}$. Each state (S, P) is composed of two "sub-states" S and S_{wt} of signatures Σ_{st} and Σ_{wt}, respectively. $\Sigma_{st} \subseteq \Sigma$ is finite and contains only the "standard" function names which are not in the reserve. Again we use $P|_{\Sigma_{st}}$ to denote the restriction of P to Σ_{st}.

Let $W_{st} \cup W_{wt}$ be a R-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:

$$G^{(S,P)}_{W_{wt}} = \{raise(t') \mid val_{(S,P)}(t) = t' \text{ for some } t \in W_{wt}\}.$$

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). As $raise(t')$ is a multiset comprehension term, its interpretation in state (S, P) will be a multiset of values. We call each element of such a multiset a *critical value* of (S, P).

LEMMA 1. *Let $\mathcal{A}$ be a RA. 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 critical values of (S, P).*

Using the key fact that by the reflective abstract state postulate the set of states of $\mathcal{A}$ is closed under *essentially* isomorphic states, this lemma can still be proven by contradiction using the same argument as in the proof of the analogous Lemma 6.2 in the sequential ASM thesis [16].

As in [12, Cor. 7.8] we now obtain for every extended state (S, P) 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. $\Delta(r_{(S,P)}, (S, P)) = \Delta(\mathcal{A}, (S, P))$.

The lengthy proof exploits the construction of critical structures, on grounds of which types are defined that correspond

to "indistinguishable" updates, which are used to show that these always appear in the same update sets. Then the existence of isolating formulae for types is shown extending corresponding results from finite model theory, and finally uses the isolating formulae to construct the rule. The courageous reader will find the proof details in [12, pp.44–52].

We proceed as in [9] by lifting the rules $r_{(S,P)}$ to obtain a reflective ASM that is behaviourally equivalent to $\mathcal{A}$.

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

$$\Delta(r_{(S,P)},(S',P')) = \Delta(\mathcal{A},(S',P')).$$

Let (S,P) and (S',P') be states of $\mathcal{A}$. We say that (S',P') is *relative $W[(S,P)]$-equivalent* to (S,P) if $G^{(S',P')}_{W_{wt}} = G^{(S,P)}_{W_{wt}}$. We further define that (S,P) and (S',P') *coincide over* $W[(S,P)]$, if $val_{(S,P)}(t) = val_{(S',P')}(t)$ for all $t \in W_{st} \cup G^{(S,P)}_{W_{wt}}$.

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 states. Δ_{st} denotes the subset of updates with function names in Σ_{st}.

COROLLARY 1. *If two states (S,P) and (S',P') of $\mathcal{A}$ are relative $W[(S,P)]$-equivalent and coincide over $W[(S,P)]$, then it follows that $\Delta_{st}(r_{(S,P)},(S',P')) = \Delta_{st}(\mathcal{A},(S',P'))$.*

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

Using terms $\varphi_{(S,P)}$ that evaluate to true exactly for states that are W-equivalent relative to $\mathcal{C}[(S,P)]$ it is straightforward to show the following lemma, which 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 state $(S_i,P_i) \in \mathcal{C}[(S,P)]$, i.e., for every state that is relative $W[(S,P)]$-equivalent to (S,P).*

Thus, for every class $\mathcal{C}([S_i,P_i])$ of 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 states which belong to some run of $\mathcal{A}$, not just for the states in the class $\mathcal{C}([S_i,P_i])$. Here is when the power of reflection becomes apparent.

Fix an arbitrary initial state (S,P) of $\mathcal{A}$. We define $\mathcal{M}$ as the *reflective* ASM machine with:

$$\mathcal{S}_{\mathcal{M}} = \{(S_i,P_i') \mid (S_i,P_i) \in \mathcal{S}_{\mathcal{A}} \text{ and } P_i' \text{ is the "self" representation of } r_{[(S_i,P_i)]}\}$$

$$\mathcal{I}_{\mathcal{M}} = \{(S_i,P_i') \mid (S_i,P_i') \in \mathcal{S}_{\mathcal{M}} \text{ and } P_i' \text{ is the "self" representation of } r_{[(S,P)]}\}$$

LEMMA 5. *For every run $(S_0,P_0),(S_1,P_1),\ldots$ of $\mathcal{A}$ and corresponding run $(S_0',P_0'),(S_1',P_1'),\ldots$ of $\mathcal{M}$ with $S_0 = S_0'$, it holds that*

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

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 $P_i|_{\Sigma_i}$ and $P_i'|_{\Sigma_i}$ have exactly the same runs. This implies our main result.

THEOREM 2. *For every RA $\mathcal{A}$ there is a behaviourally equivalent reflective ASM $\mathcal{M}$.*

3. EVOLVING CONCURRENT SYSTEMS

An asynchronous concurrent systems can be seen as a collection of agents, each equipped with an algorithm to execute. In the behavioural theory of concurrent sequential algorithms developed in [7] it was assumed that the algorithm associated with an agent is a sequential algorithm, but a hint was given that this might be generalisable to parallel algorithms exploiting the behavioural theory from [12] for parallel algorithms. We claim that it can be generalised to systems of reflective parallel algorithms.

3.1 A Concurrency Postulate

Abstracting from details of how the agents interact we may assume that there are certain shared locations that can be updated by several agents. When an agent starts one of its steps, it interacts with other agents in a given state S_j by evaluating the current values of all its input or shared locations in this state. This means that the interaction is with those agents that can write and may have written in some previous state the current values of these locations. When an agent completes its current step, it interacts again with other agents in a given state S_n, this time by writing back values to its output or shared locations, thus contributing to form the next concurrent run state S_{n+1}. It means that the interaction is with those agents that also contribute to form the next state S_{n+1} by their own simultaneous write backs in state S_n. This interpretation of the notion of "interaction" assumes that the reads and writes a process performs in a step on shared locations are executed atomically at the beginning respectively at the end of the step. This leads to the following postulate, in which the algorithms associated with the agents are characterised by the postulates for RAs.

POSTULATE 5. (CONCURRENCY POSTULATE). *An* evolving concurrent system *(ECS) is given by a set $\mathcal{A}$ of pairs $(a, alg(a))$ of agents a, each equipped with a reflective algorithm $alg(a)$. In a concurrent $\mathcal{A}$-run started in some initial state S_0, each interaction state S_n ($n \geq 0$) where some agents (those of some finite set A_n of agents) interact with each other yields a next state S_{n+1} by the moves of all agents $a \in A_n$ that happen to simultaneously complete the execution of their current $alg(a)$-step they had started in some preceding state S_j ($j \leq n$ depending on a).*

With respect to reflection and concurrency in Abstract State Machines a *reflective concurrent ASM* (rcASM) can be characterised by a family $\{\mathcal{M}_a\}_{a \in Ag}$ of reflective parallel ASMs indexed by a set of agents Ag. The semantics of a rcASM is then easily defined by *concurrent ASM runs* as defined in [7]. These runs are based on runs of the individual ASMs $\mathcal{M}_a$ without taking care how these (local) runs are defined. The extension to agents with reflective ASMs is thus straightforward.

DEFINITION 4. *Let $\mathcal{A} = \{\mathcal{M}_a\}_{a \in Ag}$ be a rcASM. A* concurrent run *of $\mathcal{A}$ is a sequence $S_0, S_1, \ldots$ of states together with a sequence $A_0, A_1, \ldots$ of subsets of Ag such that each state S_{n+1} is obtained from S_n by applying to it the updates computed by the agents $a \in A_n$ each of which started its current (internal) step by reading its input and shared locations in some preceding state S_j depending on a. The run terminates in state S_n if the updates computed by the agents in A_n are inconsistent.*

The defining condition can be expressed by the following formula where $S_{lastRead(a,n)}$ denotes the state in which a performed its reads of all monitored and shared locations it uses for the current step (so that $lastRead(a, n) \leq n$):

$$S_{n+1} = S_n + \bigcup_{a \in A_n} \Delta(\mathcal{M}_a, S_{lastRead(a,n)})$$

We say that a starts its j-th step in state S_j with $j = lastRead(a, n)$ and completes it in state S_n. Remember that $S + U$ is not defined if U is an inconsistent update set.

On these grounds it is straightforward to show again the plausibility of the concurrency postulate.

THEOREM 3. *Every rcASM $\mathcal{A} = \{\mathcal{M}_a\}_{a \in Ag}$ is an ECS satisfying the postulates for RA and the concurrency postulate.*

3.2 Characterisation Theorem for Evolving Concurrent Systems

To complete the development of the behavioural theory for ECS let us look at the converse of Theorem 3. Given an ECS $\{(a, alg(a)) \mid a \in Ag\}$ we have to construct reflective ASMs M_a for each agent $a \in Ag$. So, let $S_0, S_1, \ldots$ be the state sequence and $Ag_0, Ag_1, \ldots$ the sequence of sets of agents of any concurrent $\mathcal{A}$-run. Consider any state S_{i+1} in this run. Then (by the Concurrency Postulate) we have for some index $lastRead(a, i) \leq i$ for each $a \in Ag_i$:

$$S_{i+1} = S_i + \bigcup_{a \in Ag_i} \Delta(alg(a), S_{lastRead(a,i)} \downarrow \Sigma_{alg(a)})$$

That is, the update set defining the change from state S_i to its successor state S_{i+1} in the concurrent run is a finite union of update sets $\Delta(alg(a_j), S_{i_j} \downarrow \Sigma_{alg(a_j)})$ where $i_j = lastRead(a, i)$. For each agent $a_j \in Ag_i$ participating with a non-empty update set there exists a well-defined previous state S_{i_j} whose (possibly including monitored and shared) location values $alg(a_j)$ determine its update set. The restriction of this state to the signature Σ_j of $alg(a_j)$ is a valid state for the reflective algorithm $alg(a_j)$, and the determined update set is the unique update set defining the transition from this state $S_{i_j} \downarrow \Sigma_j$ to its next state via τ_a.

Let R_a be the set of all pairs $(S_a, \Delta(S_a))$ of states with their computed update sets where a makes a move in a concurrent $\mathcal{A}$-run, i.e. such that

- S_a is a state of $alg(a)$ and $\Delta(S_a)$ is the unique, consistent update set computed by $alg(a)$ for the transition from S_a to $\tau_a(S_a)$,
- $S_a = S_{i_j} \downarrow \Sigma_{alg(a)}$ for some state S_{i_j} in some non-terminated run of $\mathcal{A}$, in which $alg(a)$ computes an update set that contributes to the definition of a later state in the same run.

The following lemma has actually been proven in the characterisation proof for reflective parallel algorithms.

LEMMA 6. *For each agent $a \in A$ there exists a reflective ASM rule $\mathcal{M}_a$ such that for all $(S_a, \Delta(S_a)) \in R_a$ we get $\Delta(S_a) = \Delta(\mathcal{M}_a, S_a)$, i.e. the updates the ASM $\mathcal{M}_a$ yields in state S_a are exactly the updates in $\Delta(S_a)$ determined by $alg(a)$.*

So any given concurrent $\mathcal{A}$-run $S_0, S_1, \ldots$ together with $Ag_0, Ag_1, \ldots$ is indeed a concurrent $\mathcal{M}$-run of the crASM $\mathcal{M} = \{(a, \mathcal{M}_a) \mid a \in Ag\}$. This proves Theorem 4.

THEOREM 4. *Each concurrent algorithm denoted as $\mathcal{A} = \{(a, alg(a)) \mid a \in Ag\}$ as stipulated by the* Concurrency Postulate *can be simulated step-by-step by a concurrent reflective ASM $\mathcal{M} = \{(a, \mathcal{M}_a) \mid a \in Ag\}$.*

4. REASONING ABOUT EVOLVING CONCURRENT SYSTEMS

In the previous two sections we have seen that rcASMs capture ECSs. Therefore, in order to support rigorous logical inferences to prove desirable properties of rcASMs it suffices to develop a logic for rcASMs, which is what this section will address.

4.1 A Logic for Non-Deterministic ASMs

In [42] we developed a complete logic for so-called DB-ASMs based on previous work in [41]. With respect to the logic for ASMs developed by Stärk and Nachen [37] the logic makes explicit use of meta-finite states, solves the problem of non-determinism, which was considered as a hard open problem, and added multiset-based synchronisation terms. We then observed in [12] that meta-finite states were also present in parallel ASMs, so the logic actually captures non-deterministic parallel ASMs.

Thus, states of a non-deterministic, parallel ASM are meta-finite structures [14]. Each state consists of a *finite part* and a possibly infinite *algorithmic part* linked via *bridge functions*, in which actual entries in the finite part are treated merely as surrogates for the real values. A signature Υ of states comprises a sub-signature Υ_f for the finite part, a sub-signature Υ_a for the algorithmic part and a finite set $\mathcal{F}_b$ of bridge function names. The *base set* of a state S is a nonempty set of values $B = B_f \cup B_a$, where B_f is finite. Function symbols f in Υ_f and Υ_a, respectively, are interpreted as functions f^S over B_f and B_a, and the interpretation of a k-ary function symbol $f \in \mathcal{F}_b$ defines a function f^S from B_f^k to B_a. For every state over Υ the restriction to Υ_f results in a finite structure.

As in ASMs we distinguish between updatable dynamic functions and static functions which cannot be updated. Let S be a state over Υ, $f \in \Upsilon$ be a dynamic function symbol of arity n and $a_1, ..., a_n$ be elements in B_f or B_a depending on whether $f \in \Upsilon_f \cup \mathcal{F}_b$ or $f \in \Upsilon_a$, respectively. Then $(f, (a_1, ..., a_n))$ is called a *location* of S. An *update* of S is a pair (ℓ, b), where ℓ is a location and $b \in B_f$ or $b \in B_a$ depending on whether $f \in \Upsilon_f$ or $f \in \Upsilon_a \cup \mathcal{F}_b$, respectively, is the *update value* of ℓ. To simplify notation we write $(f, (a_1, \ldots, a_n), b)$ for the update (ℓ, b) with the location $\ell = (f, (a_1, \ldots, a_n))$. The interpretation of ℓ in S is called the *content* of ℓ in S, denoted by $val_S(\ell)$. An *update set* Δ is a set of updates; an *update multiset* $\ddot{\Delta}$ is a multiset

of updates. A *location operator* ρ is a multiset function that returns a single value from a multiset of values.

Let $\Upsilon = \Upsilon_f \cup \Upsilon_a \cup \mathcal{F}_b$ be a signature of states. Fix a countable set $\mathcal{X}_f$ of first-order variables, denoted with standard lowercase letters $x, y, z, \ldots$, that range over the primary finite part of the states (i.e., the finite set B_f). The set of first-order terms $\mathcal{T}_{\Upsilon,\mathcal{X}_f}$ of vocabulary Υ is defined in a similar way than in meta-finite model theory [14]. That is, $\mathcal{T}_{\Upsilon,\mathcal{X}_f}$ is constituted by the set $\mathcal{T}_{db}$ of *database terms* and the set $\mathcal{T}_a$ of *algorithmic terms*. The set of terms $\mathcal{T}_f$ is the closure of the set $\mathcal{X}_f$ of variables under the application of function symbols in Υ_f. The set of algorithmic terms $\mathcal{T}_a$ is defined inductively: If $t_1, \ldots, t_n$ are terms in $\mathcal{T}_f$ and f is an n-ary bridge function symbol in $\mathcal{F}_b$, then $f(t_1, \ldots, t_n)$ is an algorithmic term in $\mathcal{T}_a$; if $t_1, \ldots, t_n$ are algorithmic terms in $\mathcal{T}_a$ and f is an n-ary function symbol in Υ_a, then $f(t_1, \ldots, t_n)$ is an algorithmic term in $\mathcal{T}_a$; nothing else is an algorithmic term in $\mathcal{T}_a$.

Let S be a meta-finite state of signature Υ. A *valuation* or *variable assignment* ζ is a function that assigns to every variable in $\mathcal{X}_f$ a value in the base set of the finite part B_f of S. The value $val_{S,\zeta}(t)$ of a term $t \in \mathcal{T}_{\Upsilon,\mathcal{X}_f}$ in the state S under the valuation ζ is defined as usual in first-order logic. The *first-order logic of meta-finite states* is defined as the first-order logic with equality which is built up from equations between terms in $\mathcal{T}_{\Upsilon,\mathcal{X}_f}$ by using the standard connectives and first-order quantifiers. Its semantics is defined in the standard way. The truth value of a first-order formula of meta-finite states φ in S under the valuation ζ is denoted as $[\![\varphi]\!]_{S,\zeta}$.

Without loss of generality, a variable assignment ζ as previously defined for first-order variables that range over B_{db}, can be extended to first-order variables that range over B_a as well as to second-order variables that range over finite sets. We use $fr(t)$ to denote the set of (both first-order and second-order) free variables occurring in t.

Definition 5. *The set of terms in the logic for non-deterministic ASMs is constituted by the set $\mathcal{T}_f$ and the set $\mathcal{T}_a$ of* algorithmic terms *expressed as follows:*

- $x \in \mathcal{T}_f$ *for* $x \in \mathcal{X}_f$ *and* $fr(x) = \{x\}$*;*
- $\mathbf{x} \in \mathcal{T}_a$ *for* $\mathbf{x} \in \mathcal{X}_a$ *and* $fr(\mathbf{x}) = \{\mathbf{x}\}$*;*
- $f(t) \in \mathcal{T}_f$ *for* $f \in \Upsilon_f$, $t \in \mathcal{T}_f$ *and* $fr(f(t)) = fr(t)$*;*
- $f(t) \in \mathcal{T}_a$ *for* $f \in \mathcal{F}_b$, $t \in \mathcal{T}_f$ *and* $fr(f(t)) = fr(t)$*;*
- $f(t) \in \mathcal{T}_a$ *for* $f \in \Upsilon_a$, $t \in \mathcal{T}_a$ *and* $fr(f(t)) = fr(t)$*;*
- $\rho_x(t \mid \varphi(x, \bar{y})) \in \mathcal{T}_a$ *for a location operator* $\rho \in \Lambda$*, a formula* $\varphi(x, \bar{y})$ *of the logic (see Definition 6 below),* x *a variable in* $\mathcal{X}_f$*,* $\bar{y}$ *a tuple of arbitrary variables, and* $t \in \mathcal{T}_a$*.*

In the last line we require $fr(t) \subseteq fr(\varphi(x, \bar{y})) = \{x_i \mid x_i = x \text{ or } x_i \text{ appears in } \bar{y}\}$ *and* $fr(\rho_x(t|\varphi(x, \bar{y}))) = fr(\varphi(x, \bar{y})) - \{x\}$*.*

We use the notion *ρ-term* for a term $\rho_x(t|\varphi(x, \bar{y}))$ and *pure term* for a term that does not contain ρ-terms, i.e., a term that does not contain any formulae. ρ-terms are built upon formulae; on the other hand they can also be used for constructing formulae.

Definition 6. *The formulae of the logic for non-deterministic ASMs are those generated by the following grammar:*

$$\begin{aligned}\varphi, \psi \ ::= {} & s = t \mid s_a = t_a \mid \neg\varphi \mid \varphi \wedge \psi \mid \forall x(\varphi) \mid \forall \mathbf{x}(\varphi) \mid \forall M(\varphi) \\ & \mid \forall X(\varphi) \mid \forall \mathcal{X}(\varphi) \mid \forall \ddot{X}(\varphi) \mid \forall \ddot{\mathcal{X}}(\varphi) \mid \forall F(\varphi) \mid \forall G(\varphi) \\ & \mid upd(r, X) \mid upm(r, \ddot{X}) \mid M(s, t_a) \mid X(f, t, t_0) \\ & \mid \mathcal{X}(f, t, t_0, s) \mid \ddot{X}(f, t, t_0, t_a) \mid \ddot{\mathcal{X}}(f, t, t_0, t_a, s) \\ & \mid F(f, t, t_0, t_a, t', t'_0, t'_a, s) \\ & \mid G(f, t, t_0, t_a, t', t'_0, t'_a, s_a) \mid [X]\varphi\end{aligned}$$

where s, t and t' denote terms in $\mathcal{T}_f$, s_a, t_a and t'_a denote terms in $\mathcal{T}_a$, $x \in \mathcal{X}_f$ and $\mathbf{x} \in \mathcal{X}_a$ denote first-order variables, M, X, $\mathcal{X}$, $\ddot{X}$, $\ddot{\mathcal{X}}$, F and G denote second-order variables, r is an ASM rule, f is a dynamic function symbol in $\Upsilon_f \cup \mathcal{F}_b$, and t_0 and t'_0 denote terms in $\mathcal{T}_f$ or $\mathcal{T}_a$ depending on whether f is in Υ_f or $\mathcal{F}_b$, respectively.

In the logic, disjunction $\vee$, implication $\rightarrow$, and existential quantification $\exists$ are defined as abbreviations in the usual way. $\forall M(\varphi)$, $\forall X(\varphi)$, $\forall \mathcal{X}(\varphi)$, $\forall \ddot{X}(\varphi)$, $\forall \ddot{\mathcal{X}}(\varphi)$, $\forall F(\varphi)$ and $\forall G(\varphi)$ are second-order formulae in which M, X, $\mathcal{X}$, $\ddot{X}$, $\ddot{\mathcal{X}}$, F and G range over finite relations.

When applying forall and parallel rules, updates yielded by parallel computations may be identical. Thus, we need the multiset semantics for describing a collection of possible identical updates. This leads to the inclusion of $\mathrm{upm}(r, \ddot{X})$ and $\ddot{X}(f, t, t_0, t_a)$ in the logic. $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$ respectively state that a finite update set represented by X and a finite update multiset represented by $\ddot{X}$ are generated by a rule r. $X(f, t, t_0)$ describes that an update (f, t, t_0) belongs to the update set represented by X, while $\ddot{X}(f, t, t_0, t_a)$ describes that an update (f, t, t_0) occurs at least once in the update multiset represented by $\ddot{X}$. If (f, t, t_0) occurs n-times in the update multiset represented by $\ddot{X}$, then there are n distinct $a_1, \ldots, a_n \in B_a$ such that $(f, t, t_0, a_i) \in \ddot{X}$ for every $1 \leq i \leq n$ and $(f, t, t_0, a_j) \notin \ddot{X}$ for every a_j other than $a_1, \ldots, a_n$. We use $[X]\varphi$ to express the evaluation of φ over a state after executing the update set represented by X on the current state. The second-order variables $\mathcal{X}$ and $\ddot{\mathcal{X}}$ are used to keep track of the parallel branches that produce the update sets and multisets, respectively, in a way which becomes clear later on. Finally, we use M to denote binary second-order variables which are used to represent the finite multisets in the semantic interpretation of ρ-terms, and F and G to denote second-order variables which encode bijections between update multisets. Again, the need for these types of variables becomes clear later in the paper.

A formula of the logic is *pure* if it does *not* contain any ρ-term and is generated by the following restricted grammar:

$$\varphi, \psi \ ::= s = t \mid s_a = t_a \mid \neg\varphi \mid \varphi \wedge \psi \mid \forall x(\varphi) \mid \forall \mathbf{x}(\varphi)$$

As defined before the formulae occurring in conditional, forall and choice rules are pure formulae of this logic. A formula or a term is *static*, if it does not contain any dynamic function symbol.

In [42] a proof system for this logic was developed, for which soundness and completeness could be proven.

4.2 Reasoning about Reflection

With the logics for ASMs [37] and non-deterministic ASMs [42] in mind, what has to be changed to handle reflection.

The answer to this question is rather simple, as it only concerns rules r in the logic, which only appear in formulae of the form $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$.

In a non-reflective ASM the main rule is given as part of the specification and treated as extra-logical constant. However, in a reflective ASM the main rule is the value in a location such as *self*. Consequently, we have $val_S(self) = r_S$, i.e. the interpretation of the term *self* in a state S yields the rule that is to be applied in S.

However, for a single machine step this change is rather irrelevant, as in a reflective ASM the main rule does not change within a single step. Thus, we have to take multiple steps into account. For these introduce two additional predicates r-upd and r-upm with the following informal meaning:

- $\text{r-upd}(n, X)$ means that n steps of the reflective ASM yield the update set X, where in each step the actual value of *self* is used.
- $\text{r-upm}(n, X)$ means that n steps of the reflective ASM yield the update multiset X.

In the light of the axioms **U6** and **UM6** in [42], which actually define $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$ for sequence rules, we can inductively define axioms for r-upd and r-upm. Clearly, we have $\text{r-upd}(1, X) \leftrightarrow \mathrm{upd}(self, X)$. Analogously, define $\text{r-upm}(1, X) \leftrightarrow \mathrm{upm}(self, \ddot{X})$.

Then we further define

$$\begin{aligned}&\text{r-upd}(n+1, X) \leftrightarrow \big(\text{r-upd}(1, X) \wedge \neg\text{conUSet}(X)\big)\vee\\ &\big(\exists Y_1 Y_2(\text{r-upd}(1, Y_1) \wedge \text{conUSet}(Y_1) \wedge [Y_1]\text{r-upd}(n, Y_2)\wedge\\ &\qquad\bigwedge_{f\in\mathcal{F}_{dyn}} \forall xy(X(f, x, y) \leftrightarrow ((Y_1(f, x, y)\wedge\\ &\qquad\qquad \forall z(\neg Y_2(f, x, z))) \vee Y_2(f, x, y)))))\big)\end{aligned}$$

as well as

$$\begin{aligned}&\mathrm{upm}(n+1, \ddot{X}) \leftrightarrow \Big(\text{r-upm}(1, \ddot{X})\wedge\\ &\forall X\Big(\bigwedge_{f\in\mathcal{F}_{dyn}} \forall x_1x_2(X(f, x_1, x_2) \leftrightarrow \exists \mathtt{x}_3(\ddot{X}(f, x_1, x_2, \mathtt{x}_3)))\wedge\\ &\qquad\neg\text{conUSet}(X)\Big)\Big) \vee \Big(\exists \ddot{Y}_1\ddot{Y}_2\Big(\text{r-upm}(1, \ddot{Y}_1)\wedge\\ &\forall Y_1\Big(\bigwedge_{f\in\mathcal{F}_{dyn}} \forall x_1x_2(Y_1(f, x_1, x_2) \leftrightarrow \exists \mathtt{x}_3(\ddot{Y}_1(f, x_1, x_2, \mathtt{x}_3)))\wedge\\ &\qquad\text{conUSet}(Y_1) \wedge [Y_1]\text{r-upm}(n, \ddot{Y}_2)\Big)\wedge\\ &\bigwedge_{f\in\mathcal{F}_{dyn}} \forall x_1x_2\mathtt{x}_3\big(\ddot{X}(f, x_1, x_2, \mathtt{x}_3) \leftrightarrow (\ddot{Y}_2(f, x_1, x_2, \mathtt{x}_3)\vee\\ &\qquad(\ddot{Y}_1(f, x_1, x_2, \mathtt{x}_3) \wedge \forall y_2\mathtt{y}_3(\neg\ddot{Y}_2(f, x_1, y_2, \mathtt{y}_3))))\big)\Big)\end{aligned}$$

4.3 A Logic for Concurrent Reflective ASMs

Finally, in order to capture also concurrency we make a very simple, but also powerful observation that a concurrent ASM can always be mimicked by a non-deterministic ASM. For each agent a replace its rule r by

IF ctl = idle **THEN CHOOSE** r
OR local(r) ‖ ctl := active **ENDIF**
IF ctl = active **THEN CHOOSE** skip
OR final(r) ‖ ctl := idle **ENDIF**

In an initial state the "control-state" location ctl is set to idle. If this is the case the agent executes either immediately its rule or executes a local version of it, i.e. all updates will be written to a local copy. In the second case the control-state becomes active. If the control-state is active, the agent may either do nothing or finalise the execution by copying all updates to the shared locations and returning to an idle control state.

In doing so, the multi-steplogic sketched above for reflective, non-deterministic ASMs can be used to reason about concurrent, reflective ASMs. Details concerning this are subject to ongoing research.

5. CONCLUDING REMARKS

In this paper we first presented a behavioural theory for evolving concurrent systems (ECSs), which integrates the corresponding theories of (synchronous) parallel algorithms [12], reflective algorithms [9] and concurrent algorithms [7]. With this behavioural theory we lay the foundations for rigorous development of general adaptive concurrent systems. Due to the similarities of ASMs with other rigorous methods the theory is not restricted to the context of ASMs.

As ECSs are captured by reflective, concurrent ASMs (rcASMs) a second part of the paper was dedicated to the presentation of a logic for rcASMs that is based on a one-step logic for non-deterministic ASMs [42]. Concurrency can be mimicked by non-determinism, and reflection can be added by considering a multi-step extension, in which the so-far extra-logical rules can be replaced by rule terms that are subject to interpretation. This logic enables to formally reason about ECSs and to rigorously verify desirabe properties.

We envision that in general concurrent systems it will be desirable to capture also non-determinism or preferably randomised behaviour (see [34] for first steps in this direction). Furthermore, for rigorous development extensions to the refinement method for ASMs [6] will be necessary. These problems are addressed in ongoing research.

6. REFERENCES

[1] Abrial, J.-R. *The B-book - Assigning programs to meanings*. Cambridge University Press, 2005.

[2] Abrial, J.-R. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[3] An, W. Formal specification and analysis of asynchronous mutual exclusion algorithms. Master's thesis, JKU Linz, Austria, 2016.

[4] Blass, A., and Gurevich, Y. Abstract State Machines capture parallel algorithms. *ACM Trans. Computational Logic 4(4)* (2003), 578–651.

[5] Blass, A., and Gurevich, Y. Abstract State Machines capture parallel algorithms: Correction and extension. *ACM Transactions on Computation Logic 9*, 3 (2008).

[6] Börger, E. The ASM refinement method. *Formal Aspects of Computing 15*, 2-3 (2003), 237–257.

[7] Börger, E., and Schewe, K.-D. Concurrent Abstract State Machines. *Acta Informatica 53*, 5 (2016), 469–492.

[8] Börger, E., and Stärk, R. F. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

[9] Ferrarotti, F., Schewe, K.-D., and Tec, L. A behavioural theory for reflective sequential algorithms,

2016. submitted for publication.

[10] Ferrarotti, F., Schewe, K.-D., and Tec, L. A complete logic for non-deterministic parallel ASMs, 2016. submitted for publication.

[11] Ferrarotti, F., Schewe, K.-D., Tec, L., and Wang, Q. A new thesis concerning synchronised parallel computing – simplified parallel ASM thesis. *CoRR abs/1504.06203* (2015). see http://arxiv.org/abs/1504.06203.

[12] Ferrarotti, F., Schewe, K.-D., Tec, L., and Wang, Q. A new thesis concerning synchronised parallel computing – simplified parallel ASM thesis. *Theoretical Computer Science 649* (2016), 25–53. extended version in [11].

[13] Ferrarotti, F., Tec, L., and Turull Torres, J. M. Towards an ASM thesis for reflective sequential algorithms. In *Abstract State Machines, Alloy, B, TLA, VDM and Z (ABZ 2016)* (2016), M. Butler et al., Eds., vol. 9675 of *LNCS*, Springer, pp. 239–244.

[14] Grädel, E., and Gurevich, Y. Metafinite model theory. *Information and Computation 140*, 1 (1998), 26–81.

[15] Gurevich, Y. Evolving algebras 1993: Lipari Guide. In *Specification and Validation Methods*. Oxford University Press, 1995, pp. 9–36.

[16] Gurevich, Y. Sequential Abstract State Machines capture sequential algorithms. *ACM Trans. Computational Logic 1*, 1 (July 2000), 77–111.

[17] Harel, D., Kozen, D., and Tiuryn, J. *Dynamic Logic*. MIT Press, 2000.

[18] Hewitt, C. What is computation? Actor model versus Turing's model. In *A Computable Universe: Understanding Computation and Exploring Nature as Computation*, H. Zenil, Ed. World Scientific Publishing, 2012.

[19] Hoare, C. A. R. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[20] Kröger, F., and Merz, S. *Temporal Logic and State Systems*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[21] Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers 28*, 9 (1979), 690–691.

[22] Lamport, L. *Specifying Systems, The TLA$^+$ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[23] Lynch, N. *Distributed Algorithms*. Morgan Kaufmann, 1996. ISBN 978-1-55860-348-6.

[24] Mazurkiewicz, A. Trace theory. vol. 255 of *LNCS*, Springer, pp. 279–324.

[25] Mazurkiewicz, A. Introduction to trace theory. In *The Book of Traces*, V. Diekert and G. Rozenberg, Eds. World Scientific, Singapore, 1995, pp. 3–67.

[26] Merz, S. On the logic of TLA$^+$. *Computers and Artificial Intelligence 22*, 3-4 (2003), 351–379.

[27] Milner, R. *A Calculus of Communicating Systems*. Springer, 1982. ISBN 0-387-10235-3.

[28] Milner, R. *Communicating and Mobile Systems: The Pi-Calculus*. Springer, 1999. ISBN 9780521658690.

[29] Peterson, J. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[30] Petri, C. A. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik der Universität Bonn, 1962. Schriften des IIM Nr. 2.

[31] Riccobene, E., and Scandurra, P. Towards asm-based formal specification of self-adaptive systems. In *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference (ABZ 2014)* (2014), Y. A. Ameur and K.-D. Schewe, Eds., vol. 8477 of *Lecture Notes in Computer Science*, Springer, pp. 204–209.

[32] Roscoe, A. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[33] Schellhorn, G., Tofan, B., Ernst, G., Pfähler, J., and Reif, W. RGITL: A temporal logic framework for compositional reasoning about interleaved programs. *Annals af Mathematics and Artificial Intelligence 71* (2014), 1–44.

[34] Schewe, K.-D., Ferrarotti, F., Tec, L., and Wang, Q. Towards a behavioural theory for random parallel computing. In *Computational Models of Rationality – Essays Dedicated to Gabriele Kern-Isberner on the Occasion of Her 60th Birthday*, C. Beierle, G. Brewka, and M. Thimm, Eds., vol. 29 of *Tributes*. College Publications, 2016, pp. 365–373.

[35] Schmalz, M. *Formalizing the Logic of Event-B*. PhD thesis, ETH Zürich, 2012.

[36] Seebach, H., Nafz, F., Steghöfer, J.-P., and Reif, W. How to design and implement self-organising resource-flow systems. In *Organic Computing - A Paradigm Shift for Complex Systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer, 2011, pp. 145–161.

[37] Stärk, R. F., and Nanchen, S. A logic for abstract state machines. *Journal of Universal Computer Science 7*, 11 (2001), 980–1005.

[38] Steghöfer, J.-P. *Large-Scale Open Self-Organising Systems: Managing Complexity with Hierarchies, Monitoring, Adaptation, and Principled Design*. PhD thesis, University of Augsburg, 2014.

[39] Steghöfer, J.-P., Seebach, H., Eberhardinger, B., Huebschmann, M., and Reif, W. Combining PosoMAS method content with Scrum: Agile software engineering for open self-organising systems. *Scalable Computing: Practice and Experience 16*, 4 (2015), 333–354.

[40] The Petri nets bibliography, University of Hamburg. http://www.informatik.uni-hamburg.de/TGI/pnbib/index.html.

[41] Wang, Q. *Logical Foundations of Database Transformations for Complex-Value Databases*. Berlin, Germany: Logos-Verlag, 2010.

[42] Wang, Q., Ferrarotti, F., Schewe, K.-D., and Tec, L. A complete logic for non-deterministic database transformations. *CoRR abs/1602.07486* (2016). http://arxiv.org/abs/1602.07486.

[43] Winskel, G., and Nielsen, M. Models for concurrency. In *Handbook of Logic and the Foundations of Computer Science: Semantic Modelling*, S. Abramsky, D. Gabbay, and T. S. E. Maibaum, Eds., vol. 4. Oxford University Press, 1995, pp. 1–148.