A Logic for Reflective ASMs

Klaus-Dieter Schewe[1] and Flavio Ferrarotti[2(✉)]

[1] Zhejiang University, UIUC Institute, Haining, China
kd.schewe@intl.zju.edu.cn, kdschewe@acm.org
[2] Software Competence Center Hagenberg, Hagenberg, Austria
flavio.ferrarotti@scch.at

Abstract. Reflective algorithms are algorithms that can modify their own behaviour. Recently a behavioural theory of reflective algorithms has been developed, which shows that they are captured by reflective abstract state machines (rASMs). Reflective ASMs exploit extended states that include an updatable representation of the ASM signature and rules to be executed by the machine in that state. Updates to the representation of ASM signatures and rules are realised by means of a sophisticated tree algebra defined in the background of the rASM. In this paper the theory is taken further by an extension of the logic of ASMs to capture inferences on rASMs. The key is the introduction of terms that are interpreted by ASM rules stored in some location. We show that fragments of the logic with a fixed bound on the number of steps preserve completeness, whereas the full run-logic for rASMs becomes incomplete.

Keywords: Abstract state machine · Reflection · Logic · Tree algebra

1 Introduction

Reflection refers to the ability of an algorithm or program to modify its own behaviour. The concept is as old as computer science; it already appears in LISP [16], where programs and data are both represented uniformly as lists. General run-time and compile-time linguistic reflection in programming and database research have been investigated in general by Stemple, Van den Bussche and others in [18,19]. Recently, adaptivity and thus reflection has become a key aspect of (cyber-physical) systems [7]. Nonetheless, it is still not well understood and contains great challenges and risks. As it is hard to oversee how a system behaves after many adaptations, any uncontrolled application of reflection bears the risk of unpredictable and undesired outcomes. Thus, the challenge for rigorous methods is to enable static reasoning and verification of desired properties of reflective algorithms and systems, which requires to control an unbounded family of specifications.

The research reported in this paper has been partly funded by BMVIT, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG.

A. Raschke et al. (Eds.): ABZ 2020, LNCS 12071, pp. 93–106, 2020.
https://doi.org/10.1007/978-3-030-48077-6_7

Concerning the foundations of reflection we developed a behavioural theory of reflective sequential algorithms (RSAs) in [12] (see arXiv version in [9]), which extends and cleanses our previous sketch in [2]. The theory provides an axiomatic, language-independent definition of RSAs, defines an extension of sequential ASMs to reflective sequential ASMs (rsASMs), by means of which RSAs can be specified, and provides a proof that RSAs are captured by rsASMs. That is, rsASMs satisfy the postulates of the axiomatisation, and any RSA as stipulated by the axiomatisation can be defined by a behaviourally equivalent rsASM. The notion of *behavioural equivalence* is slightly weaker than the corresponding notion for sequential or parallel algorithms, as there is no need to require that changes to the represented algorithm are exactly the same, as long as the application of the algorithm to the core part of the structure yields the same results.

In [13] we sketched how to generalise the theory to reflective parallel algorithms [11], which requires an integration of the behavioural theory of synchronous parallel algorithms [3]. Leaving this general aspect aside the generalisation of just the reflective sequential ASMs to reflective ASMs is rather straightforward. For deterministic ASMs this was done in [10]. In a nutshell, in each step of a reflective ASM (rASM) the rule is taken from a dedicated location *self*, which uses a tree structure to represent the signature and rule, and a sophisticated tree algebra to manipulate tree values [14]. We also exploit partial updates in the form of [15] to minimise clashes that may otherwise result from simultaneously updating *self* by several parallel branches.

In this paper we address the fundamental question how desired properties of a reflective algorithm can be verified. As rASMs capture reflective algorithms, this requires extending the logic of ASMs [4,5,17]. We observe that in these logics the rules defining an ASM only enter as extra-logical constants r that are expanded in atomic formulae $[r]\varphi$ (the application of r to the current state leads to a state satisfying the formula φ), $\mathrm{upd}(r, X)$ (the rule r yields an update set X in the current state), and $\mathrm{upm}(r, \ddot{X})$ (the rule r yields an update multiset $\ddot{X}$ in the current state). In an rASM, however, the rule to be applied in the current state is stored itself in the state in a sublocation of a location *self*. We therefore explore the idea to treat r in formulae as variables that are interpreted by a rule stored in the current state. Furthermore, as reasoning about reflective algorithms only makes sense for multiple steps, we also extend the one-step ASM logic to a multiple-step logic. The precise definition of such a logic and the completeness proof for a fragment of the logic are the key contributions of this paper.

In Sect. 2 we present rASMs as extensions of ASMs. Section 3 is dedicated to the introduction of the logic of ASMs, which follows our previous work in [4]. The core of the paper is Sect. 4, where we formally develop the extension of the logic dealing with reflection and investigate completeness. We conclude with a brief summary and outlook in Sect. 5.

2 Reflective Abstract State Machines

We assume general familiarity with ASMs as defined in [1]. The extension to reflective ASMs requires to define a background structure that covers trees and operations on them, a dedicated variable *self* that takes as its value a tree representation of an ASM signature and rule, and the extension of rules by partial updates. Due to space limitations our presentation must be terse—nevertheless the details are given in [9,10,12]. Note that the omitted details include the sophisticated tree algebra defined for the representation of rules and the access to them. We use some of its operators, but they can be correctly understood from the context.

Let Σ be an ASM signature, i.e. a set of function symbols. Partial assignments are defined as follows: Whenever $f \in \Sigma$ has arity n and op is an operator of arity $m+1$, t_i $(i = 1, \ldots, n)$ and t'_i $(i = 1, \ldots, m)$ are terms over Σ, then $f(t_1, \ldots, t_n) \Leftarrow^{op} t'_1, \ldots, t'_m$ is a rule. The informal meaning is that we evaluate the terms as well as $f(t_1, \ldots, t_n)$ in the current state S, then apply op to $\mathrm{val}_S(f(t_1, \ldots, t_n)), \mathrm{val}_S(t'_1), \ldots, \mathrm{val}_S(t'_m)$ and assign the resulting value v to the location $(f, (\mathrm{val}_S(t_1), \ldots, \mathrm{val}_S(t_n)))$. Conditions for compatibility and the collapse of an update multiset into an update set have been elaborated in detail in [15].

For the dedicated location storing the self-representation of an ASM it is sufficient to use a single function symbol *self* of arity 0. Then in every state S the value $\mathrm{val}_S(\mathit{self})$ is a complex tree comprising two subtrees for the representation of the signature and the rule, respectively. That is, in the tree structure we have a root node o labelled by `self` with exactly two successor nodes, say o_0 and o_1, labelled by `signature` and `rule`, respectively. So we have $o \prec_c o_0$, $o_0 \prec_s o_1$ and $o \prec_c o_1$, where $\prec_c$ and $\prec_s$ denote, respectively, the child and sibling relationships. The subtree rooted at o_0 has as many children $o_{00}, \ldots, o_{0k}$ as there are function symbols in the signature, each labelled by `func`. Each of the subtrees rooted at o_{oi} takes the form $\texttt{func}\langle\texttt{name}\langle f\rangle\texttt{arity}\langle n\rangle\rangle$ with a function name f and a natural number n. The subtree rooted at o_1 represents the rule of a sequential ASM as a tree.

The inductive definition of trees representing rules is rather straightforward. For instance, an assignment rule $f(t_1, \ldots, t_n) := t_0$ is represented by a tree of the form $\texttt{update}\langle\texttt{func}\langle f\rangle\texttt{term}\langle t_1 \ldots t_n\rangle\texttt{term}\langle t_0\rangle\rangle$, and a partial assignment rule $f(t_1, \ldots, t_n) \Leftarrow^{op} t'_1, \ldots, t'_m$ is represented by a tree of the form $\texttt{partial}\langle\texttt{func}\langle f\rangle\texttt{func}\langle op\rangle\texttt{term}\langle t_1 \ldots t_n\rangle\texttt{term}\langle t'_1 \ldots t'_m\rangle\rangle$.

The *background of an rASM* is defined by a background class $\mathcal{K}$ over a background signature V_K. It must contain an infinite set *reserve* of reserve values and an infinite set Σ_{res} of reserve function symbols, the equality predicate, the undefinedness value *undef*, and a set L of labels `self`, `signature`, `rule`, `func`, `name`, `arity`, `update`, `term`, `if`, `bool`, `par`, `let`, `partial`. The background class must further define truth values and their connectives, tuples and projection operations on them, natural numbers and operations on them, trees in T_L and tree operations, and the function $\mathbf{I}$, where $\mathbf{I}x.\varphi$ denotes the unique x satisfying condition φ.

If B is a base set, then an *extended base set* is the smallest set B_{ext} containing B that is closed under adding function symbols in the reserve Σ_{res}, natural numbers, the terms $\mathbb{T}$ with respect to B and Σ_{res}, and terms of the tree algebra defined over Σ_{res} with labels in L as defined above. Furthermore, we use $\hat{\mathbb{T}}_{ext}$ to denote the union of the set $\mathbb{T}_{ext}$ of terms with Σ_{ext} and the set of rules.

The background must further provide functions: *drop* : $\hat{\mathbb{T}}_{ext} \to B_{ext}$ and *raise* : $B_{ext} \to \hat{\mathbb{T}}_{ext}$ for each base set B and extended base set B_{ext}, and a derived *extraction function* $\beta : \mathbb{T}_{ext} \to \bigcup_{n\in\mathbb{N}} \mathbb{T}^n$, which assigns to each term defined over the extended signature Σ_{ext} and the extended base set B_{ext} a tuple of terms in $\mathbb{T}$ defined over Σ and B.

A *reflective ASM* (rASM) $\mathcal{M}$ comprises an (initial) signature Σ containing a 0-ary function symbol *self*, a background as defined above, and a set $\mathcal{I}$ of initial states over Σ closed under isomorphisms such that any two states $I_1, I_2 \in \mathcal{I}$ coincide on *self*. Furthermore, $\mathcal{M}$ comprises a state transition function τ on states over extended signature Σ_S with $\tau(S) = S + \Delta_{r_S}(S)$, where the rule r_S is defined as $raise(rule(\mathrm{val}_S(self)))$ over the signature $\Sigma_S = raise(signature(\mathrm{val}_S(self)))$.

In this definition we use extraction functions *rule* and *signature* defined on the tree representation of a sequential ASM in *self*. These are simply defined as $signature(t) = subtree(\mathbf{I}o.root(t) \prec_c o \wedge label(o) = \texttt{signature})$ and $rule(t) = subtree(\mathbf{I}o.root(t) \prec_c o \wedge label(o) = \texttt{rule})$.

3 The Logic of Abstract State Machines

We now look briefly into a simplified version of the logic of non-deterministic ASMs as defined in [4]. The simplification concerns the distinction between db-terms and algorithmic terms that is necessary, if explicit meta-finite states are considered. Here we just consider a single uniform signature Σ, so terms are defined in the usual way. However, we have to keep in mind that rASMs have a rich set of operators in their background that are used to build terms. Furthermore, as we are dealing with non-determinism there is a need to consider also ρ-terms of the form $\rho_v(t \mid \varphi)$, where ρ is a multiset operator defined in the background, φ is a formula, t is a term, and v is a variable. A *pure term* is defined as a term that does not contain any sub-term which is a ρ-term.

In order to define formulae inductively we extend the set of first-order variables with a countable set of second-order (relation) variables of arity r for each $r \geq 1$.

1. If s and t are terms, then $s = t$ is a formula.
2. If $t_1, \ldots, t_r$ are terms and X is a second-order variable of arity r, then $X(t_1, \ldots, t_r)$ is a formula.
3. If r is a rule and X is a second-order variable of arity 3, then $\mathrm{upd}(r, X)$ is a formula.
4. If r is a rule and $\ddot{X}$ is a second-order variable of arity 4, then $\mathrm{upm}(r, \ddot{X})$ is a formula.
5. If φ and ψ are formulae and x is a first-order variable, then $\neg\varphi$, $\varphi \vee \psi$ and $\forall x(\varphi)$ are formulae.

6. If φ is a formula and X is a second-order variable, then $\forall X(\varphi)$ is a formula.
7. If φ is a formula and X is a second-order variable of arity 3, then $[X]\varphi$ is a formula.

Note that we use second-order variables of arity 3 and 4 to capture update sets and update multisets, respectively.

The semantics of the logic is defined by Henkin structures. A *Henkin prestructure* S is a state of signature Σ with base set B extended with a new universe D_n of n-ary relations for each $n \geq 1$, where $D_n \subseteq \mathcal{P}(B^n)$.

Variable assignments ζ into a Henkin prestructure S are defined as usual: $\zeta(x) \in B$ for each first-order variable x, and $\zeta(X) \in D_n$ for each second-order variable X of arity n.

Then the interpretation of a term in a Henkin prestructure S with a variable assignment ζ is defined as usual; for ρ-terms $t = \rho_v(t' \mid \varphi)$ we have $val_{S,\zeta}(t) = \rho(\{\!\{val_{S,\zeta[v\mapsto a_i]}(t') \mid a_i \in B \text{ and } [\![\varphi]\!]_{S,\zeta[v\mapsto a_i]} = true\}\!\})$.

We extend this interpretation to formulae. For a second-order variable X of arity 3 we abuse the notation by writing $val_{S,\zeta}(X) \in \Delta(r, S, \zeta)$ meaning that there is a set $U \in \Delta(r, S, \zeta)$ such that $(f, a_0, a_1) \in U$ iff $(c_f^S, a_0, a_1) \in val_{S,\zeta}(X)$. Analogously, for a second-order variable $\ddot{X}$ of arity 4 we write $val_{S,\zeta}(\ddot{X}) \in \ddot{\Delta}(r, S, \zeta)$ meaning that there is a multiset $\ddot{U} \in \Delta(r, S, \zeta)$ such that $(f, a_0, a_1) \in \ddot{U}$ with multiplicity n iff there are exactly $b_1, \ldots, b_n$ pairwise different values such that $(c_f^S, a_0, a_1, b_i) \in val_{S,\zeta}(X)$ for every $1 \leq i \leq n$. If φ is a formula, then its truth value on S under ζ (denoted as $[\![\varphi]\!]_{S,\zeta}$) is defined by the following rules:

- If φ is of the form $s = t$, then $[\![\varphi]\!]_{S,\zeta} = \begin{cases} true & \text{if } val_{S,\zeta}(s) = val_{S,\zeta}(t) \\ false & \text{otherwise} \end{cases}$.
- If φ is of the form $X(t_1, \ldots, t_r)$, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} true & \text{if } (val_{S,\zeta}(t_1), \ldots, val_{S,\zeta}(t_n)) \in val_{S,\zeta}(X) \\ false & \text{otherwise} \end{cases}.$$

- If φ is of the form $\mathrm{upd}(r, X)$, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} true & \text{if } val_{S,\zeta}(X) \in \Delta(r, S, \zeta) \\ false & \text{otherwise} \end{cases}.$$

- If φ is of the form $\mathrm{upm}(r, \ddot{X})$, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} true & \text{if } val_{S,\zeta}(\ddot{X}) \in \ddot{\Delta}(r, S, \zeta) \\ false & \text{otherwise} \end{cases}.$$

- If φ is of the form $\neg\psi$, then $[\![\varphi]\!]_{S,\zeta} = \begin{cases} true & \text{if } [\![\psi]\!]_{S,\zeta} = false \\ false & \text{otherwise} \end{cases}$.

– If φ is of the form $\alpha \vee \psi$, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} \mathit{true} & \text{if } [\![\alpha]\!]_{S,\zeta} = \mathit{true} \text{ or } [\![\psi]\!]_{S,\zeta} = \mathit{true} \\ \mathit{false} & \text{otherwise} \end{cases}.$$

– If φ is of the form $\forall x(\psi)$, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} \mathit{true} & \text{if } [\![\psi]\!]_{S,\zeta[x\mapsto a]} = \mathit{true} \text{ for all } a \in B \\ \mathit{false} & \text{otherwise} \end{cases}.$$

– If φ is of the form $\forall X(\psi)$, where X is a second-order variable of arity n, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} \mathit{true} & \text{if } [\![\psi]\!]_{S,\zeta[X\mapsto R]} = \mathit{true} \text{ for all } R \in D_n \\ \mathit{false} & \text{otherwise} \end{cases}.$$

– If φ is of the form $([X]\psi)$, then

$$[\![\varphi]\!]_{S,\zeta} = \begin{cases} \mathit{false} & \text{if } \zeta(X) \text{ represents an update set } U \\ & \text{such that } U \text{ is consistent and } [\![\psi]\!]_{S+U,\zeta} = \mathit{false} \\ \mathit{true} & \text{otherwise} \end{cases}.$$

For a sentence φ to be valid in the given Henkin semantics, it must be true in all Henkin prestructures. This is a stronger requirement than saying that φ is valid in the standard Tarski semantics. A sentence that is valid in Tarski semantics is true in those Henkin prestructures, for which each universe D_n is the set of all relations of arity n.

The universes D_n of the Henkin prestructures should not be arbitrary collections of n-ary relations. Thus, it is reasonable to restrict our attention to some collections of n-ary relations that we can define, i.e. we restrict our attention to Henkin structures.

A *Henkin structure* is a Henkin prestructure S that is closed under definability, i.e. for every formula φ, variable assignment ζ and arity $n \geq 1$, we have that $\{\bar{a} \in A^n \mid [\![\varphi]\!]_{S,\zeta[a_1\mapsto x_1,\ldots,a_n\mapsto x_n]} = \mathit{true}\} \in D_n$.

The main result in [4] states that the logic for ASMs defined here is complete with respect to Henkin semantics.

4 Reasoning About Reflection

Let us now investigate the extension of the logic above to handle reflection. The main difference of rASMs to ordinary ASMs is that in each step a different rule r is applied, and this rule is part of the current state. In the one-step logic of ASMs described in the previous section a rule is treated as a fixed extra-logical constant appearing only in formulae of the form $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$, and the meaning of these formulae depends on the actual rule r.

4.1 Extension of the Logic of ASMs

In an rASM $val_S(self)$ is a tree value t and $rule(t)$ (defined at the end of Sect. 2) is the subtree representing the actual rule of the rASM in state S. Then $raise(rule(\mathrm{val}_S(self)))$ is the rule r_S of the rASM in state S, or phrased differently, we obtain this rule by interpretation of the term

$$\mathbf{therule} \;=\; raise(subtree(\mathbf{I}o.root(self) \prec_c o \wedge label(o) = \texttt{rule}).$$

That is, the only extension to the logic required to capture reflection is the treatment of the first argument of $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$ as a term that is then evaluated in the state S. If the result is not a rule, these formulae remain undefined.

However, for a single machine step this extension is rather irrelevant, as in an rASM the main rule does not change within a single step. Thus, we have to take multiple steps into account. For these we 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.
- $\text{r-upm}(n, \ddot{X})$ means that n steps of the reflective ASM yield the update multiset $\ddot{X}$.

To be more precise, X and $\ddot{X}$ in predicates $\text{r-upd}(n, X)$ and $\text{r-upd}(n, \ddot{X})$ are the *union* of the n update sets and n updates multisets, respectively, yielded by the reflective ASM in n steps.

Clearly, we have $\text{r-upd}(1, X) \leftrightarrow \mathrm{upd}(\mathbf{therule}, X)$, and analogously, $\text{r-upm}(1, X) \leftrightarrow \mathrm{upm}(\mathbf{therule}, \ddot{X})$. For the generalisation to arbitrary values of n we exploit the definition of $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$ for sequence rules to inductively define axioms for r-upd and r-upm. We further need the definition of consistent update sets in the logic:

$$\mathrm{conUSet}(X) \;\equiv \bigwedge_{c_f \in \mathcal{F}_{dyn}} \forall xyz\Big(\big(X(c_f, x, y) \wedge X(c_f, x, z)\big) \rightarrow y = z\Big)$$

for the set $\mathcal{F}_{dyn}$ of constants representing the dynamic function symbols in Σ. Then we can use $\mathrm{con}(r, X)$ to expresses that X represents one of the possible update sets generated by a rule r and that X is consistent:

$$\mathrm{con}(r, X) \equiv \mathrm{upd}(r, X) \wedge \mathrm{conUSet}(X).$$

We further define

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

as well as

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

4.2 Completeness

Let $\mathcal{L}^{(r)}_{asm}$ denote the logic of rASMs resulting from these extensions using **therule** and predicates r-upd(n,X) and r-upm(n,X) for arbitrary n. Let $\mathcal{L}^{r}_{asm}$ denote the further extended logic of rASMs, in which in addition quantification over n is permitted. Let us call $\mathcal{L}^{(r)}_{asm}$ the *multi-step logic* of rASMs, and $\mathcal{L}^{r}_{asm}$ the *run logic* of rASMs.

Even without updating the rule in every step it is obvious that the run logic $\mathcal{L}^{r}_{asm}$ subsumes a full dynamic logic over runs of ASMs. As such it is impossible to achieve completeness.

Theorem 1. *The run logic $\mathcal{L}^{r}_{asm}$ of rASMs is incomplete.*

Concerning the multi-step logic $\mathcal{L}^{(r)}_{asm}$ of rASMs the situation is not so obvious. We may continue a sublogic $\mathcal{L}^{(r,n)}_{asm}$ using a fixed value of n and formulae of the form r-upd(m,X) and r-upm(m,X) with fixed $m\leq n$. For such a sublogic we can extend the completeness result of the logic of ASMs using similar arguments.

Theorem 2. *For each $n\in\mathbb{N}$ the bounded fraction $\mathcal{L}^{(r,n)}_{asm}$ of the multi-step logic $\mathcal{L}^{(r)}_{asm}$ of rASMs is complete.*

The remaining part of this section is dedicated to prove this key result.

First note that every subformulae of the form r-upd(m,X) and of the form r-upm(m,X) that occurs in a $\mathcal{L}^{(r,n)}_{asm}$-formulae can be replaced by their corresponding definitions above. This is possible, because we have only bounded finite values for $m=1\ldots n$ to consider.

Thus, the axioms and rules of the derivation system remain the same as for the logic of ASMs [4,5]. Starting point is the natural formalism L_2 as defined in [6] for the relational variant of second-order logic on which the logic is based.

L_2 uses the usual axioms and rules for first-order logic, with quantifier rules applying to second-order variables as well as first-order variables, and with the stipulation that the range of the second-order variables includes at least all the relations definable by the formulae of the language. A deductive calculus for L_2 is obtained by augmenting the axioms and inference rules of first-order logic as follows:

- $\exists X \forall v_1, \ldots, v_k (X(v_1, \ldots, v_k) \leftrightarrow \varphi)$, where $k \geq 1$, $v_1, \ldots, v_k$ are first-order variables, and X is a k-ary second-order variable that does not occur freely in the formula φ.
- $\forall X(\varphi) \rightarrow \varphi[Y/X]$, provided the arity of X and Y coincides.
- $\dfrac{\psi \rightarrow \varphi[Y/X]}{\psi \rightarrow \forall X(\varphi)}$, provided Y is not free in ψ.

In addition to these axioms and rules and standard axioms and rules for first-order logic with equality, the logic $\mathcal{L}_{asm}^{(r,n)}$ comprises the following:

- The axioms for $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, X)$. Since here we do not need to consider explicit meta-finite states, these axioms are a simplified version of Axioms U1–U7 and Axioms Ü1–Ü7 in Section 7.2 and 7.3 in [4], respectively. For instance, Axiom U1 which states that X represents an update set yielded by the assignment rule $f(t) := s$ iff it contains exactly one update and this update is $((f, t), s)$, can be written as:

  U1: $\mathrm{upd}(f(t) := s, X) \leftrightarrow X(c_f, t, s) \wedge \forall zxy(X(z, x, y) \rightarrow z = c_f \wedge x = t \wedge y = s)$

- The distribution axiom and the necessitation rule from the axiom system **K** of modal logic, and *modus ponens*, which allow us to derive all modal properties that are valid in Kripke frames.
- The axiom $\neg\mathrm{conUSet}(X) \rightarrow [X]\varphi$ asserting that if an update set X is not consistent, then there is no successor state obtained from applying X to the current state—thus $[X]\varphi$ is interpreted as true for any formula φ.
- The axiom $\neg[X]\varphi \rightarrow [X]\neg\varphi$ describing the deterministic accessibility relation in terms of $[X]$.
- The Barcan axiom $\forall v([X]\varphi) \rightarrow [X]\forall v(\varphi)$, where v is a first-order or second-order variable.
- Axioms $\varphi \wedge \mathrm{upd}(r, X) \rightarrow [X]\varphi$ and $\mathrm{con}(r, X) \wedge [X]\varphi \rightarrow \varphi$ for static and pure φ asserting that the interpretation of static and pure formulae is the same in all states.
- The frame axiom $\mathrm{conUSet}(X) \wedge \forall z(\neg X(c_f, x, z)) \wedge f(x) = y \rightarrow [X]f(x) = y$ and the update axiom $\mathrm{conUSet}(X) \wedge X(c_f, x, y) \rightarrow [X]f(x) = y$ asserting the effect of applying an update set.
- The axiom $\mathrm{upm}(r, X) \rightarrow \exists Y(\mathrm{upd}(r, Y))$ stating that if a rule r yields an update multiset, then it also yields an update set.
- The restricted axiom of universal instantiation $\forall v(\varphi(v)) \rightarrow \varphi[t/v]$, if φ is pure or t is static, t is a term free for v in $\varphi(v)$.

– The rule of universal generalisation $\dfrac{\psi \to \varphi[v'/v]}{\psi \to \forall v(\varphi)}$ if v' is not free in ψ.
– The axiom

$$\exists X(\mathrm{upd}(\mathbf{seq}\ r_1\ r_2\ \mathbf{endseq}, X) \wedge [X]\varphi) \leftrightarrow$$
$$\exists X_1(\mathrm{upd}(r_1, X_1) \wedge [X_1]\exists X_2(\mathrm{upd}(r_2, X_2) \wedge [X_2]\varphi)).$$

from dynamic logic asserting that executing a sequence rule is equivalent to executing its sub-rules sequentially.
– The extensionality axiom

$$r_1 \equiv r_2 \to (\exists X_1.\mathrm{upd}(r_1, X_1) \wedge [X_1]\varphi) \leftrightarrow \exists X_2.\mathrm{upd}(r_2, X_2) \wedge [X_2]\varphi.$$

For the proof of completeness we proceed in the same way as for the corresponding completeness proof for the logic of ASMs in [4]. First for operators defined in the background, in particular the multiset functions used in ρ-terms, are treated as standard non-axiomatised functions. This allows us to assume without loss of generality that formulae do not contain ρ-terms.

Then we turn formulae into variants of formulae of first-order logic with types. For this we create a modified signature Σ^T, which contains the function symbols from Σ, a unary relation symbol T_n for each $n \geq 1$, an $(n+1)$-ary relation symbol E_n for each $n \geq 1$, and unary relation symbols T_0 and T_r. With these we proceed as follows:

1. Turn formulae $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, X)$ into formulae of the form $T_r(x) \to \mathrm{upd}(x, X)$ and $T_r(x) \to \mathrm{upm}(x, X)$, where $T_r(x)$ asserts that x is a tree term representing a rule.
2. Bring all remaining atomic formulae into the form $v_1 = v_2$, $f(v_2) = v_1$ or $X(v_1, \ldots, v_n)$.
3. Eliminate all modal operators expressing them by means of the formula $\mathrm{conUSet}(X)$.
4. Replace each atomic (second-order) formula of the form $X(t_1, \ldots, t_n)$ by $E_n(t_1, \ldots, t_n, X)$, and relativise quantifiers over individuals using T, and quantifiers over n-ary relations in D_n for some $n \geq 1$ to T_n.

The main difference to the similar reduction applied in [4] is that subformulae $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, X)$ cannot be completely eliminated. However, by using T_r and tree terms we turn these formulae into first-order formulae with types. Then the axioms for $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, X)$ have to be adapted to this modification as well. In the case of upd, we define a new axiom that replaces Axioms U1–U7 and has the form

$$\mathrm{upd}(x, X) \leftrightarrow \varphi_{U1}(x, X) \vee \cdots \vee \varphi_{U7}(x, X),$$

where $\varphi_{U1}, \ldots, \varphi_{U7}$ are modified versions of the formulae in the right-hand side of Axioms U1–U7, respectively. In particular, φ_{U1} can be defined as follows:

$$\exists x_0 x_1 x_2 x_3 x_f x_t x_s \big((x_0 = \mathbf{I}o.\mathit{root}(x) \prec_c o \wedge \mathit{label}(o) = \texttt{update}) \wedge$$

$$label(x_1) = \texttt{func} \wedge label(x_2) = \texttt{term} \wedge label(x_3) = \texttt{term} \wedge$$

$$x_0 \prec_c x_1 \wedge x_0 \prec_c x_2 \wedge x_0 \prec_c x_3 \wedge x_2 \prec_s x_3 \wedge x_1 \prec_c x_f \wedge x_2 \prec_c x_t \wedge x_3 \prec_c x_s$$

$$\exists y_f y_t y_s (y_f = val_S(raise(x_f)) \wedge y_t = val_S(raise(x_t)) \wedge y_s = val_S(raise(x_s)) \wedge$$

$$\wedge X(y_f, y_t, y_s) \wedge \forall z_f z_t z_s (X(z_f, z_f, z_s) \rightarrow z_f = y_f \wedge z_t = y_t \wedge z_s = y_s)))$$

Note that for simplicity we have assumed, w.l.o.g. (see [4] among others), that the arity of the functions in the update rules is 1.

Due to space limitations, we leave the definition of the remaining formulae $\varphi_{U2}, \ldots, \varphi_{U7}$ as a simple exercise to the reader. Likewise the definition of a new axiom that replaces Axioms Ü1–Ü7 is also left as an easy exercise to the reader.

Lemma 1. *A formula φ of $\mathcal{L}_{asm}^{(r,n)}$ is true in a Henkin prestructure S iff the transformed formula φ^* is true in a first-order structure S^* over Σ^T that is uniquely determined by S.*

The first direction of Lemma 1, i.e., if an $\mathcal{L}_{asm}^{(r,n)}$-formula φ is true in S, then φ^* is true in S^*, can be proven by structural induction. We only need to apply the transformation described above to each of the cases in the definition of the set of $\mathcal{L}_{asm}^{(r,n)}$-formulae and then check that the resulting first-order formulae is satisfied by the corresponding state S^*. Likewise, the second direction of Lemma 1 can be proven by structural induction on the definition of first-order formulae, in this case using the inverse of the transformation described above. We omit these proofs since both are quite long, but technically straightforward.

Thus, if φ^* is valid, then φ is true in all Henkin structures. Note that the converse does not always hold. For instance $\exists x(x = x)$ is true in all Henkin structures (since by definition the domain of S is not empty), but $\exists x(T_0(x) \wedge (x = x))$ is not valid. In general, *not every* Σ^T-structure is an S^* structure for some Henkin Σ-structure S. However, if a Σ^T-structure S^* satisfies the following properties, then it corresponds to a Henkin structure S (cf. [6]):

1. Σ-correctness:
   - $T_r(c)$ for nullary function symbols *self* $\in \Sigma$,
   - $T_0(c)$ for all nullary function symbols $c \in \Sigma$ other than *self*, and
   - $\bigwedge_{1 \leq i \leq n} T_0(x_i) \rightarrow T(f(x_1, \ldots, x_n))$ for every $f \in \Sigma$.
2. Non-emptiness: $\exists x(T_0(x) \vee T_r(x))$.
3. Disjointness:
   - $T_i(x) \rightarrow \neg T_j(x)$ for $i, j \geq 0$ with $i \neq j$,
   - $T_r(x) \rightarrow \neg T_i(x)$ and $T_i(x) \rightarrow \neg T_r(x)$ for $i \geq 0$.
4. Elementhood: $E_n(x_1, \ldots, x_n, y) \rightarrow T_n(y) \wedge (T_0(x_1) \vee T_r(x_1)) \wedge \cdots \wedge (T_0(x_n) \vee T_r(x_n))$ for $n \geq 1$.
5. Extensionality: $T_n(x) \wedge T_n(y) \wedge \forall \bar{z}(E_n(\bar{z}, x) \leftrightarrow E_n(\bar{z}, y)) \rightarrow x = y$ for $n \geq 1$.
6. Comprehension: $\exists y \forall \bar{x}(E_n(\bar{x}, y) \leftrightarrow \psi)$ for $n \geq 1$ and y non-free in ψ.

Lemma 2. *If A is a first-order structure of signature Σ^T which satisfies properties 1–6 above and $sub(A)$ is the sub-structure of A induced by the elements of $\bigcup_{n\geq 0}(T_n)^A \cup (T_r)^A$, then for some Henkin structure S of signature Σ, $sub(A)$ is the structure S^* determined by S.*

Proof. Given A with domain $dom(A)$, we define S as follows:

- $dom(S) = (T_0)^A \cup (T_r)^A$ is the base set (of individuals) of S.
- For each $n \geq 1$, the universe D_n of n-ary relations consists of the sets $\{\bar{a} \in (dom(S))^n \mid (E_n)^A(\bar{a}, s)\}$ for all $s \in (T_n)^A$.
- The interpretation of function symbols $f \in \Sigma$ is the same as in A but restricted to arguments from $dom(S)$.

By the Σ-correctness, non-emptiness and comprehension properties of A, we get that S is a Henkin structure.

We claim that $sub(A)$ is isomorphic to S^* via function $g : dom(S^*) \to dom(sub(A))$ where

$$g(x) = \begin{cases} x & \text{if } x \in (T_0)^{S^*} \cup (T_r)^{S^*} \\ \{\bar{a} \in ((T_0)^{S^*} \cup (T_r)^{S^*})^n \mid (E_n)^{S^*}(\bar{a}, x)\} & \text{if } x \in (T_n)^{S^*} \text{ for } n \geq 1 \end{cases}$$

First, we note that g is well defined by the disjointness property and by the fact that, by definition of S and S^*, every element x in $dom(S^*)$ is in $\bigcup_{n\geq 0}(T_n)^{S^*} \cup (T_r)^{S^*}$. That g is surjective follows from the definition of S^* from A and the fact that $dom(sub(A))$ is the restriction of $dom(A)$ to $dom(S^*)$. By the extensionality property, we get that g is injective. By definition we get that g preserves the function symbols in Σ as well as the relation symbols T_n for every $n \geq 0$. Finally, for every $n \geq 1$, we get that g preserves E_n by the elementhood property. □

Let Ψ be the set of formulae listed under properties 1–6 above, we obtain the following Henkin style completeness theorem:

Theorem 3. *An $\mathcal{L}^{(r,n)}_{asm}$-formula φ is true in all Henkin structures iff φ^* is derivable in first-order logic from Ψ (i.e., iff $\Psi \vdash \varphi^*$).*

Proof. Assume that $\Psi \vdash \varphi^*$, and let S be a Henkin structure. Then $S^* \models \Psi$ and therefore $S^* \models \varphi^*$. By Lemma 1, we get that $S \models \varphi$.

Conversely, assume that φ is true in all Henkin structures. Towards showing $\Psi \models \varphi^*$, let us assume that $A \models \Psi$, and let $sub(A)$ be its substructure generated by the elements of $\bigcup_{n\geq 0}(T_n)^A \cup (T_r)^A$. Then by Lemma 2, $sub(A) = S^*$ for some first-order structure S^* determined by a Henkin structure S. Since by assumption we have that $S \models \varphi$, it follows from Lemma 1 that $S^* \models \varphi^*$ and therefore $sub(A) \models \varphi^*$. But each quantifier in φ^* is relativised to $(T_n)^A$ for some $n \geq 1$, and then we also have that $A \models \varphi^*$. We have shown that $\Psi \models \varphi^*$, and then, by the completeness theorem of first-order logic, we get that $\Psi \vdash \varphi^*$. □

It is easy to see that the proof system that we have described earlier in this section is sound. Thus, if φ is a formula derivable in $\mathcal{L}_{asm}^{(r,n)}$, then φ is true in all Henkin structures. It is then immediate from Theorem 3 that φ^* is derivable in first-order logic from Ψ. On the other hand, via an easy but lengthy induction on the length of the derivations, we get the following.

Lemma 3. *φ^* is derivable in first-order from Ψ iff φ is derivable in $\mathcal{L}_{asm}^{(r,n)}$.*

Theorem 3 and Lemma 3 immediately imply that $\mathcal{L}_{asm}^{(r,n)}$ is complete.

5 Conclusion

We have shown before that reflective algorithms are captured by reflective abstract state machines (rASMs), which exploit extended states that include an updatable representation of the main ASM rule to be executed by the machine in that state. Updates to the representation of ASM signatures and rules are realised by means of a sophisticated tree algebra. This enables the rigorous specification of reflective algorithms and thus adaptive systems and is one step in the direction of controlling the risk associated with systems that can change their own behaviour.

In this paper we made another step in this direction by providing an extension of the logic of ASMs to rASMs. For this we replaced extra-logical constants representing rules by terms that are subject to interpretation in the current state. As reasoning about reflective algorithms only makes sense for multiple steps, we also extend the one-step ASM logic to a multiple-step logic, and prove that for a sublogic with the number of steps bound to a fixed constant we preserve the completeness of the logic, whereas the logic in general will be incomplete.

By providing such a logic we show that it is possible to reason *statically* over specifications that are highly *dynamic* and even *unbounded* in the sense that the behaviour of the system after a sequence of adaptations is not known at all at the time the system is specified. This is of tremendous importance for the application of rigorous methods to truly adaptive systems. Even more, by showing that fragments of the logic that deal with bounded sequences of steps are still complete we even enable tool support for such reasoning.

The use of the logic in an extension of proof obligations for the refinement of rASMs in the line of [8] will be the next step in our research.

References

1. Börger, E., Stärk, R.: Abstract State Machines. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36498-6
2. Ferrarotti, F., Schewe, K.-D., Tec, L.: A behavioural theory for reflective sequential algorithms. In: Petrenko, A.K., Voronkov, A. (eds.) PSI 2017. LNCS, vol. 10742, pp. 117–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74313-4_10

3. 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)
4. Ferrarotti, F., Schewe, K.-D., Tec, L., Wang, Q.: A complete logic for Database Abstract State Machines. Logic J. IGPL **25**(5), 700–740 (2017)
5. Ferrarotti, F., Schewe, K.-D., Tec, L., Wang, Q.: A unifying logic for nondeterministic, parallel and concurrent Abstract State Machines. Ann. Math. Artif. Intell. **83**(3–4), 321–349 (2018)
6. Leivant, D.: Higher order logic. In: Handbook of Logic in Artificial Intelligence and Logic Programming, Deduction Methodologies, vol. 2, pp. 229–322. Oxford University Press (1994)
7. Riccobene, E., Scandurra, P.: Towards ASM-based formal specification of self-adaptive systems. ABZ 2014. LNCS, vol. 8477, pp. 204–209. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43652-3_17
8. Schellhorn, G.: Verification of ASM refinements using generalized forward simulation. J. UCS **7**(11), 952–979 (2001)
9. Schewe, K., Ferrarotti, F.: Behavioural theory of reflective algorithms I: reflective sequential algorithms. CoRR, abs/2001.01873 (2020)
10. Schewe, K.-D.: Concurrent reflective Abstract State Machines. In: 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (SYNASC 2017), pp. 30–35. IEEE Computer Society (2017)
11. Schewe, K.-D.: Behavioural theory of reflective algorithms II: reflective parallel algorithms (2019, under review)
12. Schewe, K.-D., Ferrarotti, F.: Behavioural theory of reflective algorithms I: reflective sequential algorithms (2019, under review)
13. Schewe, K.-D., Ferrarotti, F., Tec, L., Wang, Q., An, W.: Evolving concurrent systems: behavioural theory and logic. In: Proceedings of the Australasian Computer Science Week Multiconference, (ACSW 2017), pp. 77:1–77:10. ACM (2017)
14. Schewe, K.-D., Wang, Q.: XML database transformations. J. UCS **16**(20), 3043–3072 (2010)
15. Schewe, K.-D., Wang, Q.: Partial updates in complex-value databases. In: Information and Knowledge Bases XXII, Frontiers in Artificial Intelligence and Applications, vol. 225, pp. 37–56. IOS Press (2011)
16. 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)
17. Stärk, R., Nanchen, S.: A logic for abstract state machines. J. Univ. Comput. Sci. **7**(11), 952–979 (2001)
18. Stemple, D., et al.: Type-safe linguistic reflection: a generator technology. In: 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
19. Van den Bussche, J., Van Gucht, D., Vossen, G.: Reflective programming in the relational algebra. J. Comput. Syst. Sci. **52**(3), 537–549 (1996)