Monodic Abstract State Machines and temporal verification

Michael Fisher and Alexei Lisitsa

Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, U.K.
{M.Fisher, A.Lisitsa}@csc.liv.ac.uk

Abstract. Since its introduction by Yuri Gurevich, the Abstract State Machine (ASM) approach has proved particularly successful in specifying abstract algorithms, realistic programming languages, distributed computations, and a variety of different types of hardware. In this paper, we pursue the goal of automatic deductive verification for certain classes of ASM. In particular, we base our work on a translation of general ASMs to full first-order temporal logic. While such a logic is, in general, not finitely axiomatisable, recent work has identified a fragment, termed the monodic fragment, that is finitely axiomatisable. Thus, in this paper, we define a class of monodic ASMs whose semantics in terms of temporal logic fits within the monodic fragment. This, together with recent work on clausal resolution methods for monodic fragments, allows us to carry out temporal verification of properties of monodic ASMs.

1 Introduction

Abstract State Machines (ASMs), formerly known as Evolving Algebras [17], were introduced by Yuri Gurevich in 1988 as the key part of an abstract formal method for specifying computational systems. The underlying idea, which provides ASMs with much of their flexibility, is that states of an arbitrary computation are represented by general *many-sorted first-order structures*. The vocabularies of these structures may be freely chosen in order to model any particular algorithm at an appropriate level of abstraction. (This flexibility contrasts with many standard models, such as Turing machines or finite state automata, where the alphabets used are fixed.) The process of computation is then modelled by the evolution of states according to given rules; on being 'fired', these rules change the structure produced by re-interpreting its basic functions.

Since its introduction, the ASM approach has proved to be very successful in specifying a variety of different types of software and hardware, including abstract algorithms, realistic programming languages, and distributed computations [1]. This success provides empirical evidence supporting the Abstract State Machine Thesis [17], which asserts that *every* algorithm is behaviourally equivalent to an ASM and so may be simulated, step-by-step, by that machine. For example, in [18] and [2], this thesis was established for very general formalisations of the notions of *sequential* and *parallel* algorithms, respectively.

The combination of generality, varying levels of abstractness and strong formal foundations has led to the ASM approach beginning to be applied in significant industrial applications [19, 20, 21].

However, while specification and implementation techniques using ASMs have been studied in depth, there has been relatively little work on the verification of ASMs. Research in this area has, so far, mainly concerned either human-assisted proofs of correctness [16], dealing with ASM representations in a relatively manual way, or algorithmic verification of ASMs [3, 9, 37, 38], based on model checking techniques [5].

In this paper, we pursue the goal of automatic deductive verification for certain classes of ASM. In particular, we base our work on a translation of general ASMs to full first-order temporal logic [11]. While such a logic is, in general, not finitely axiomatisable, recent work has identified a fragment, termed the *monodic* fragment, that is finitely axiomatisable [24]. Thus, in this paper, we define a class of monodic ASMs whose semantics in terms of temporal logic fits within the monodic fragment. This, together with recent work on clausal resolution methods [12, 13] for monodic fragments [6, 7], allows us to carry out temporal verification of properties of monodic ASMs.

1.1 Structure of the Paper

We begin, in Sections 2 and 3 respectively, with descriptions of the ASM approach and the variety of temporal logic we use. In Section 4, we define how general ASMs can be translated into formulae in first-order temporal logic, and establish formal properties of such a translation. Given that a translation of general ASMs to temporal logic is possible, we next consider, in Section 5, what class of ASMs translate to *monodic* first-order temporal logic. We define the class of *monodic ASMs* that capture this property, thus allowing systems specified in this class to be translated into a temporal fragment amenable to automatic (or at least semi-automatic) verification. In Section 6, we consider two applications: one concerning a distributed consensus problem; the other concerning cach coherence protocols. In Section 7 we compare our work with related work of A.Nowack. Finally, in Section 8, we provide concluding remarks and discuss future work.

2 Abstract State Machines

Definition 1
An Abstract State Machine, $\mathcal{M}$, can be defined as a tuple $\mathcal{M} = \langle \sigma, TR, IC, \Psi_0 \rangle$ where:

- σ is a *many-sorted signature*, i.e. a finite collection of function names, together with their sorts; σ is a disjoint union of three (non necessarily nonempty) parts: $\sigma = \sigma_s \cup \sigma_d \cup \sigma_e$, which consist of *static*, *dynamic* and *external* names, respectively. We assume the static part, σ_s includes constants for all elements of any finite sort.
- TR is a finite set of *transition rules*, i.e. the *program* of $\mathcal{M}$, where a *transition rule* is an expression of the form
  if $g(\bar{x})$ **then** $U_1(\bar{x}), \dots U_m(\bar{x})$
  where $g(\bar{x})$ is a first-order expression (a *guard*) of the given signature over variables $\bar{x}$ and $U_i(\bar{x})$ is an expression of the form $f(t_1(\bar{x}), \dots, t_k(\bar{x})) = t(\bar{x})$ where $f \in \sigma_d$ and $\bar{x}$ are all free variables of the update, although any term $t_1, \dots t_k, t$ is not required to contain all of $\bar{x}$;

– *IC* is a set of *integrity constraints*, with each being a first-order sentence within $\sigma_s \cup \sigma_e$;
– Ψ_0 is a first-order sentence in the vocabulary σ describing initial condition.

The idea underlying the separation of σ into three parts is as follows. The static names, i.e. those from σ_s are assumed to not change their interpretations during the computation. The interpretations of dynamic names (σ_d) may change during the computation by the application of transition rules from TR. External names (σ_e) serve to model the interaction with the environment and their interpretations are not subject to change by transition rule applications, but nevertheless their interpretations may be different in different states. It is natural to consider the evaluation of an external function as being carried out by an oracle (environment).

Note 1. Our definition is somewhat more general than common one, where instead of the sentence Ψ_0 an initial first-order structure S_0 is assumed.

Note 2. We do not allow dynamic functions into integrity constraints. There are two reasons for that. First, that would made the definition of operational semantics more complicated. Second, one can model any possible effects of such constraints by appropriate extension of transition rules. We feel, however, that in some cases constraints including dynamic functions are more natural and we are going to consider these elsewhere.

Definition 2 A *computation (run)* of an ASM $\mathcal{M}$ is a (possibly infinite) sequence of its states $S_0, \ldots S_k, \ldots$, where

– $S_0 \models \Psi_0$, i.e. initial state satifies initial condition.
– domain of each S_i coincides with that of S_0;
– the interpretations of all functional symbols from σ_s in each S_i coincide with those in S_0;
– the interpretation of all functional symbols from σ_d in S_{i+1} is obtained from the state S_i by applying of *all* transition rules, subject to the restrictions that every S_i satisfies all the integrity constraints from *IC* and the updates are consistent in S_i. The *application* of a transition rule within a given state S_i of $\mathcal{M}$ means simultaneously executing the updates $U_1(\bar{x}), \ldots, U_m(\bar{x})$ over the state S_{i+1} whenever the guard, $g(\bar{x})$ is satisfied, i.e. when $S_i \models g(\bar{x})$. The *consistency* of updates in a state S means no two different updates are to update *differently* the interpretation of same function on the same argument.
– the interpretation of all functional symbols from σ_e may be arbitrary, provided every S_i satisfies all the integrity constraints from *IC*.

Note 3. If, at some state S, the updates are inconsistent it is said that M crashes. For example, the rules **if** b **then** $f(3) = 5$ and **if** b **then** $f(3) = 7$ would lead to the crash of the machine in a state S such that $S \models b$.

Note 4. We consider basic and restricted forms of ASM. The most notable omission is *import* instructions, allowing new elements to be imported to the domain during of computation [17]. In addition, we do not consider other important extensions of ASM such as *non-deterministic choice*, *object-orientation*, or *distribution* mechanisms [1].

Definition 3 An ASM rule is said to be normal if, and only if, it consists of at most one update in its right-hand side. An ASM program is normal if, and only if, it consists only normal rules.

Proposition 1. *Any ASM program can be transformed into an equivalent normal ASM program.*

Proof. Simply replace any rule **if** $g(\bar{x})$ **then** $U_1(\bar{x}), \ldots, U_m(\bar{x})$ with the set of rules
$\{$ **if** $g(\bar{x})$ **then** $U_i(\bar{x}) \mid i = 1 \ldots m\}$.

3 Temporal Logic

The representation of dynamic activity via temporal formalisms is used in a wide variety of areas within Computer Science and Artificial Intelligence, for example temporal databases, program specification, system verification, agent-based systems, robotics, simulation, planning, knowledge representation, and many more. While we are not able to describe all these aspects here, the interested reader should see, for example, [30, 31, 5, 39, 28].

3.1 First-Order Temporal Logic (FOTL)

Here, we concentrate on one very popular variety, namely discrete linear temporal logic, which has an underlying model of time isomorphic to the Natural Numbers (i.e. an infinite sequence with distinguished initial point) and is also linear, with each moment in time having at most one successor. The infinite and linear constraints ensure that each moment in time has *exactly* one successor.

The language $\mathcal{TL}$ of the first order temporal logic over the natural numbers is constructed in the standard way from a classical (non-temporal) first order language $\mathcal{L}$ and a set of future-time temporal operators '$\Diamond$' (*sometime*), '$\Box$' (*always*), '$\bigcirc$' (*in the next moment*), '$\mathcal{U}$'(*until*).

Formulae in $\mathcal{TL}$ are interpreted in *first-order temporal structures* of the form $\mathfrak{M} = \langle D, I \rangle$, where D is a non-empty set, the *domain* of $\mathfrak{M}$, and I is a function associating with every moment of time $n \in \mathbb{N}$ an interpretation of predicate, function and constant symbols of $\mathcal{L}$ over D. First-order (nontemporal) structures corresponding to each point of time will be denoted $\mathfrak{M}_n = \langle D, I(n) \rangle$. Intuitively, the interpretations of $\mathcal{TL}$-formulae are sequences of first-order structures, or *states* of $\mathfrak{M}$, such as $\mathfrak{M}_0, \mathfrak{M}_1, \ldots, \mathfrak{M}_n \ldots$. An *assignment* in D is a function $\mathfrak{a}$ from the set $\mathcal{L}_{\mathbf{v}}$ of individual variables of $\mathcal{L}$ to D. If P is a predicate symbol then $P^{I(n)}$ (or simply P^n if I is undersood) is the interpretation of P in the state $\mathfrak{M}_n$. We require that (individual) variables and constants of $\mathcal{TL}$ are *rigid*, that is neither assignments nor interpretations of constants depend on the state in which they are evaluated. On the other hand we allow both predicates and functional symbols to be *flexible*. Flexibility of functions, which is not a very common assumption in the semantics of FOTL, is very convenient for direct modelling of ASM rules.

The *truth-relation* $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$ (or simply $n \models^{\mathfrak{a}} \varphi$, if $\mathfrak{M}$ is understood) in the structure $\mathfrak{M}$ for the assignment $\mathfrak{a}$ is defined inductively in the usual way under the following semantics of temporal operators:

$n \models^{\mathfrak{a}} \bigcirc\varphi$ iff $n+1 \models^{\mathfrak{a}} \varphi$;
$n \models^{\mathfrak{a}} \Diamond\varphi$ iff there is $m \geq n$ such that $m \models^{\mathfrak{a}} \varphi$;
$n \models^{\mathfrak{a}} \Box\varphi$ iff $m \models^{\mathfrak{a}} \varphi$ for all $m \geq n$;
$n \models^{\mathfrak{a}} \varphi\,\mathcal{U}\psi$ iff there is $m \geq n$ such that $m \models^{\mathfrak{a}} \psi$ and
$k \models^{\mathfrak{a}} \varphi$ for every $n \leq k < m$;

A formula φ is said to be *satisfiable* if there is a first-order structure $\mathfrak{M}$ and an assignment $\mathfrak{a}$ such that $\mathfrak{M}_0 \models^{\mathfrak{a}} \varphi$. If $\mathfrak{M}_0 \models^{\mathfrak{a}} \varphi$ for every structure $\mathfrak{M}$ and for all assignments then φ is said to be *valid*.

3.2 Monodic FOTL

While *first-order temporal logic* (FOTL) is very expressive and, in general, highly undecidable, recent results have identified recursively enumerable fragments of FOTLs [24].

Definition 4 A first-order temporal formula is said to be **monodic** if, and only if, any subformula with its main connective being a temporal operator has at most one free variable.

The monodic fragment of FOTL has appealing properties. It is axiomatizable [40] and many of its subfragments, such as the two-variable or monadic cases, are decidable. Decidability of these fragments holds also for the case of semantics where only temporal structures over *finite domains* are allowed. However, these good properties hold mainly for the case when equality is not allowed. Adding equality makes even very restricted fragments non-axiomatizable [8]. A notable exception is the packed monodic fragment with equality, which is decidable [23].

The following definitions are adopted from [23] (we use a different set of temporal connectives, all of which are definable in the variant of [23]).

Definition 5 Let σ be a vocabulary with at most constants and relation symbols. A formula γ is said to be a *packing guard* if γ is a conjunction of atomic and existentially-quantified atomic formulae (possibly equalities) such that any distinct free variables of γ co-occur free in some conjunct of γ.

Definition 6 The monodic packed fragment of predicate temporal logic in the vocabulary σ as above consists of the following formulae:

- Any atomic formula (which can be an equality) is a monodic packed formula.
- Boolean combinations of monodic packed formulae are monodic packed.
- If γ is a packing guard, φ is monodic packed formula, every free variable of φ is free in γ and $\bar{y}$ is a tuple of variables, then $\exists\bar{y}(\gamma \wedge \varphi)$ is monodic packed formula.
- If φ is monodic packed formula with at most one free variable, then $\Box\varphi$ and $\bigcirc\varphi$ are monodic packed formulae.

3.3 Temporal Tools

The widespread use of temporal logics has led to the development of key techniques for utilising temporal specifications, such as model checking and deductive verification [35, 13]. While the former has restrictions in terms of the size of models considered, the latter has problems in terms of worst-case complexity. For propositional (discrete, linear) temporal logics, the decision procedure is already PSPACE [36]. In spite of this, a range of useful tools have been developed, with our clausal resolution approach [13] having been shown to be particularly effective [26, 27].

For general (again, discrete and linear) FOTL, incompleteness is a problem. While there have been formalisations of proof mechanisms, for example infinitary sequent systems, proof in such complex logics will require significant user interaction. We have developed a prover, based on the λClam proof planning system [34] which incorporates techniques from the clausal resolution method in order to aid proof search [4].

The breakthrough concerning monodic FOTL is now leading to the development of improved proof methods. We have extended our clausal resolution approach to this fragment [6, 7], and have also provided usable implementations [25]. We believe this to be a most productive route for temporal verification in the future.

4 From ASM to FOTL

In this paper, FOTL provides the key tool for formalising the execution of ASMs. In particular, as the operational semantics of an ASM is defined in terms of runs (sequences, or partially-ordered sets, of states), these can naturally be seen as the models of FOTL.

$$
\begin{aligned}
[\![\mathcal{M}]\!] &= [\![\Psi_0]\!]^{\sigma}_{initial} \wedge [\![\sigma]\!]_{sorts} \wedge [\![\sigma]\!]_{static} \wedge [\![TR]\!]^{\sigma}_{rules} \wedge \\
&\quad \wedge [\![IC]\!]^{\sigma}_{integrity} \wedge [\![TR]\!]^{\sigma}_{frame} \\
[\![\Psi_0]\!]^{\sigma}_{initial} &= \Box(\mathbf{start} \Rightarrow [\![\Psi_0]\!]^{\sigma}_{fo}) \\
[\![\sigma]\!]_{static} &= \Box \bigwedge_{f\in\sigma_s} \forall \bar{x}, y\, Persists(f(\bar{x}) = y) \\
[\![\{r_1, r_2, \ldots, r_n\}]\!]^{\sigma}_{rules} &= \bigwedge_{i=1}^{n} [\![r_i]\!]^{\sigma}_{rule} \\
[\![\{c_1, c_2, \ldots, c_n\}]\!]^{\sigma}_{integrity} &= \Box \bigwedge_{i=1}^{n} [\![c_i]\!]^{\sigma}_{fo}
\end{aligned}
$$

Fig. 1. ASM → FOTL Translation Functions

4.1 Translation

We assume in this section that an ASM program is given in a normal form. In Fig. 1, we present translation functions which effectively provide a semantics in terms of first-order temporal logic for an ASM, '$\mathcal{M}$'. In the Figure and below we use the notation $Persists(\varphi(\bar{x}))$ to denote $\varphi(\bar{x}) \rightarrow \bigcirc\varphi(\bar{x})$. Four semantic functions remain undefined and require further explanation.

First-Order Translation. The $[\![\ ]\!]^{\sigma}_{fo}$ function simply takes the many-sorted first order description given as an argument and produces a first-order description to be used in the temporal semantics. In the case of general ASMs considered here, this is simply the identity function. Later, however, we will see that some modification of the form of the first-order sentences can be useful when translating from ASM notation to FOTL formulae.

Sort constraints. The $[\![\sigma]\!]_{sorts}$ function takes the vocabulary σ and produces ($\square$-closed) one-sorted first-order description of all sort constraints assumed for many-sorted functional symbols. For finite sorts, it also contains sentences enumerating explicitely the elements of these sorts.

Translation of Rules. Before we present the general translation of rules, we will consider an example. Translating the ASM transition rule, **if** $b(x)$ **then** $f(3,k(x)) = 2$, we have to take into account that the term $k(x)$ and the update $f(3,k(x)) = 2$ according to the semantics of ASM rules, should be evaluated in different time slices. So, a straighforward translation $\square\forall x(b(x) \rightarrow \bigcirc f(3,k(x)) = 2)$ would be incorrect. The correct translation is

$$\square\forall x\forall v[(b(x) \wedge (v = k(x)) \rightarrow \bigcirc f(3,v) = 2]$$

In general, given any rule r: **if** $g(\bar{x})$ **then** $f(t_1(\bar{x}),\ldots,t_k(\bar{x})) = t(\bar{x})$, we define the formula

$$\Phi_r(\bar{x}, v_1, \ldots, v_k) \leftrightarrow g(\bar{x}) \wedge \bigwedge_{i=1\ldots k} (v_i = t_i(\bar{x})) \tag{1}$$

Then we define the translation $[\![r]\!]$ of r as

$$\square[\forall \bar{x}, \bar{v}, z(\Phi_r(\bar{x}, \bar{v}) \wedge z = t(\bar{x}) \rightarrow \bigcirc f(\bar{v}) = z)]$$

Frame Conditions. In addition to the transition rules and integrity constraints that are explicitly part of the ASM notation, there are implicit *frame conditions*. In essence, these conditions aim to ensure that any variable not affected by an update (or integrity constraint) retains its previous value. Thus, the formula produced by $[\![TR]\!]^{\sigma}_{frame}$ attempts to capture this persistence. The approach we follow in generating this formula is based on the simple idea provided by Reiter [33], namely that all the potential updates and guards within transition rules are examined and new formulae are added to describe the behaviour if one or more of the guards are *not* satisfied.

Consider as an example ASM program with the following transition rules:

if $b(x)$ **then** $f(3,k(x)) = 2$
if $c(x)$ **then** $f(x,5) = 7$
if d **then** $f(4,m(x)) = 3$

Given some state s we are interested in identifying those evaluations val, of variables v and w, for which $g(v,w)$ will not change its value after executing the above rules in s. It is clear that this is the case precisely when $g(v,w)|val$ is not an instance of a left-hand side of any of the updates under substitution which makes the corresponding guard true. Now, for the above transition rules, we have the following frame condition (where variables v, w, and z are universally quantified and the whole formula is assumed to be under "always" operator):

$$\begin{pmatrix} \neg\exists x(v = 3 \wedge w = k(x) \wedge b(x)) \\ \wedge \\ \neg\exists x(x = v \wedge w = 5 \wedge c(x)) \\ \wedge \\ \neg\exists x(v = 4 \wedge w = m(x) \wedge d) \end{pmatrix} \Rightarrow Persists(f(v,w) = z)$$

So, more generally, we define $[\![TR]\!]^{\sigma}_{frame}$ as follows. For a function symbol f define $I_f = \{i | r_i \in TR$ and r_i updates $f\}$. Then, let $[\![TR]\!]^{\sigma,f}_{frame}$ be

$$\Box\forall\bar{v},z[(\bigwedge_{i\in I_f} \neg\exists\bar{x}\Phi_{r_i}(\bar{x},\bar{v})) \rightarrow Persists(f(\bar{v}) = z)]$$

where Φ_r is defined above (1). Finally, we define $[\![TR]\!]^{\sigma}_{frame}$ as $\bigwedge_{f\in\sigma_d} [\![TR]\!]^{\sigma,f}_{frame}$

4.2 Correctness of the Translation

Here, we give the general statement of correctness of the translation of ASMs into full (i.e. including equality) FOTL. This is based on the strong correspondence between the execution sequences of ASMs, and the models of FOTL. The operational semantics of sequential ASMs is defined in terms of runs i.e. sequences of first-order structures, which allows to consider the *infinite* runs as the models for discrete, linear first-order logic. The restriction to infinite runs[1] is essential since we consider FOTL over an infinite flow of time isomorphic to ω.

Theorem 1 (Correctness of Translation) The set $Runs(\mathcal{M})$ of infinite runs of an ASM, $\mathcal{M}$, is exactly the set of models $Mod([\![\mathcal{M}]\!])$ of the formula $[\![\mathcal{M}]\!]$, as defined in Fig. 1 and Section 4.1

Proof [OUTLINE]. Given $\mathcal{M} = \langle\sigma, TR, IC, \Psi_0\rangle$, then Fig. 1 defines

$$[\![\mathcal{M}]\!] = [\![\Psi_0]\!]^{\sigma}_{initial} \wedge [\![\sigma]\!]_{static} \wedge [\![TR]\!]^{\sigma}_{rules} \wedge [\![IC]\!]^{\sigma}_{integrity} \wedge [\![TR]\!]^{\sigma}_{frame}$$

Consider an arbitrary infinite sequence $\gamma = S_0, S_0, \ldots S_k, \ldots$ of first-order structures in the vocabulary σ. Then we have

[1] In order to deal with finite runs of ASMs, ending by reaching some end condition (not by crashing!) one may consider FOTL over finite time flows. Or, alternatively, one may formally extend the finite runs into infinite ones by adding obvious rules to ASM inforcing the machine to loop over the last state forever.

- $S_0 \models \Psi_0$ if, and only if, $S_0 \models [\![\Psi_0]\!]^{\sigma}_{initial}$ by definition of $[\![\Psi_0]\!]^{\sigma}_{initial}$;
- the interpretation of any of the functional symbols from the static part σ_s of the vocabulary is the same in all S_i if, and only if, $\gamma \models_{FOTL} [\![\sigma]\!]_{static}$ by definition of $[\![\sigma]\!]_{static}$;
- let $[\![TR]\!]^{\sigma}_{rules} = \Box STEP_{rules}$ and $[\![TR]\!]^{\sigma}_{frame} = \Box STEP_{frame}$,
  then for all i, S_{i+1} is obtained from S_i by applying all the transition rules from TR if, and only if, $(S_i, S_{i+1}) \models_{FOTL} STEP_{rules} \wedge STEP_{frame}$, by definitions of $[\![TR]\!]^{\sigma}_{rules}$, $[\![TR]\!]^{\sigma}_{frame}$, of application of transition rules and of $\models_{FOTL}$;
- For all i, S_i satisfies all the constraints from $IC = \{c_1, \ldots, c_n\}$ if, and only if, $S_i \models \bigwedge_{i=1}^{n} [\![c_i]\!]^{\sigma}_{fo}$ by definitions of $[\![c_i]\!]^{\sigma}_{fo}$ and $[\![IC]\!]^{\sigma}_{integrity}$.

It follows that $\gamma \in Runs(\mathcal{M})$ if, and only if, $\gamma \in Mod([\![\mathcal{M}]\!])$. □

4.3 Verification of ASM Translations

Given an ASM program $\mathcal{M}$ one may use the above translation for verification of this program as follows: express correctness conditions by a first-order temporal formula ψ and then check whether the implication $[\![\mathcal{M}]\!] \rightarrow \psi$ is valid FOTL formula. In general, of course we have no decision, or even semi-decision procedure to check validity (or satisfiability) of such formulae. But, if the resulting formula turns out to be in a 'good' (e.g. monodic) fragment one can apply decision procedures, or theorem proving, for verification. Otherwise, one can apply heuristic methods to show its validity. Some such methods, derived from the resolution procedures for monodic fragments, and incomplete in general are presented in [4]. In what follows we describe restrictions on ASM programs that ensure their temporal translations are in monodic fragments.

5 From ASM to Monodic FOTL: monodic ASMs

In this section we are looking for restrictions of ASM programs which ensure that their temporal translations fall within "good" monodic fragments. A quick glance at the defined translations reveals that non-monodic formulae may appear both in the translations of transition rules and in frame conditions.

Definition 7 A functional symbol f of a sort $S_1 \times \ldots \times S_k \rightarrow S$ within the ASM vocabulary is said to be *finitely valued* if, and only if, the sort S is finite.

Definition 8 A normal transition rule, **if** $g(\bar{x})$ **then** $f(t_1(\bar{x}), \ldots, t_k(\bar{x})) = t(\bar{x})$ is called *monodic* if, and only if,

1. At most one free variable ranging over an infinite sort occurs both in the guard and in the update (i.e. the free variable is shared);
2. The main functional symbol f of an update has at most one infinite input sort;
3. All functional symbols in the rule are finitely valued.

Definition 9 An ASM program is called *monodic* if, and only if,

- it consists of only normal monodic rules, and
- static part σ_s of the vocabulary consists of only finitely valued functional symbols of arities at most 1, and
- either of the following two conditions holds:
  a) immediate subterms of inifinite sort in any of main functional symbols of the updates are variables (not constants or complex terms), or
  b) all integrity constraints and initial conditions are the formulae in the packed fragment.

Examples. Let S_i be an infinite sort, S_f be a finite sort and a be a constant of sort S_f.

1. Assuming $f : S_i \times S_i \to S_f$, then the rule **if** true **then** $f(x,x) = a$ is not monodic (condition 2 does not hold)
2. Assume now that $f : S_i \times S_f \to S_f$, $k : S_i \times S_f \to S_f$. Then the rule **if** true **then** $f(x,k(x)) = a$ *is* monodic.

5.1 Translation of an monodic ASM program.

Before we formulate a general statement on the translation of monodic ASM programs into temporal logic, it is instructive to consider an example of such a translation. Take the transition rule **if** true **then** $f(x,k(x)) = a$ of the previous example and consider an ASM program $\mathcal{M}$ which contains this as its only rule. Assume also $S_f = \{a,b\}$. Then the translation $[\![TR]\!]^{\sigma}_{rules}$ of the rule is $\Box\forall x,v(x = v \to \bigcirc(f(v,k(v)) = a)$ and frame condition $[\![TR]\!]_{frame}$ is $\forall v,w,z(\neg\exists x(v = x \wedge w = k(x)) \to Persists(f(v,w) = z))$ which, after obvious simplification, become $\Box\forall x\bigcirc(f(x,k(x)) = a)$ and

$$\forall v,w,z(w \neq k(v) \to Persists(f(v,w) = z))$$

Notice that the formula expressing the frame condition is not monodic: *Persists* is applied to the subformula with *three* free variables, which destroys monodicity. Further, both above translations contain functional symbols and equality, which also makes it problematic to be in any decidable, or axiomatizable, fragment of FOTL. However, replacing functional symbols with corresponding predicates and, further, using the finiteness conditions imposed by definition of a monodic ASM program, makes it possible to translate these formulae to the good monodic fragment, preserving the “meaning” of the original ASM program.

To replace functional symbols, we introduce new relational symbols (i.e. boolean valued functional symbols)
$val_f : S_i \times S_f \times S_f \to Bool$;
$val_k : S_i \times S_f \to Bool$
Then we can rewrite $[\![TR]\!]^{\sigma}_{rules}$ into $\forall x\Box\bigcirc(\exists u(val_k(x,u) \wedge val_f(x,u,a)))$ which is now monodic and without functional symbols. Further, using the finiteness of the sort S_f we can remove the existensial quantifier:

$$[\![TR]\!]^{*}_{rules} = \forall x\Box\bigcirc\begin{bmatrix}((val_k(x,a) \wedge val_f(x,a,a)) \vee \\ (val_k(x,b) \wedge val_f(x,b,a))\end{bmatrix}$$

Now, rewrite the frame condition into

$$\forall v, w, z(\neg val_k(v,w) \rightarrow Persists(f(v,w) = z))$$

Notice, that this is still a non-monodic formula. Denote it by $\forall v, w\ \Psi(v,w)$. Further, using finiteness of S_f we can remove variables v and w and quantifiers upon them, making the resulting formula monodic:

$$[\![TR]\!]^*_{frame} = \Psi(a,a) \wedge \Psi(a,b) \wedge \Psi(b,a) \wedge \Psi(b,b)$$

To finish this translation we have to specify the *functionality* property of newly introduced relational symbols. The corresponding formula $Fun_{k,f}$:

$$\forall x, y((val_k(x,a) | val_k(x,b)) \wedge (val_f(x,y,a) | val_f(x,y,b)))$$

has to be added to the integrity constrains $[\![IC]\!]_{integrity}$. Here $|$ stands for "exclusive OR".

Notice that $[\![TR]\!]^*_{rules} \wedge [\![TR]\!]^*_{frame} \wedge Fun_{k,f}$ is a monodic formula without functional symbols and equality. Notice also that it models closely the behaviour of the initial translation $[\![TR]\!]_{rules} \wedge [\![TR]\!]_{frame}$: the sets of their temporal models are the same modulo renaming $f \mapsto val_f$ and $k \mapsto val_k$. Now we can formulate a general statement

Theorem 1. *Let $\mathcal{M}$ be monodic ASM program. Then its temporal translation $[\![\mathcal{M}]\!]$ can be further translated into monodic temporal formula $[\![\mathcal{M}]\!]^*$ which:*

- *does not contain functional symbols;*
- *either does not contain equality, or contains it in a very restricted way (i.e., is in the packed fragment of monodic logic with equality); and*
- *has the same set of temporal models as $[\![\mathcal{M}]\!]$ modulo some renaming of functional symbols into relational ones.*

Proof [OUTLINE]. Given a translation $[\![\mathcal{M}]\!]$ of an ASM program $\mathcal{M} = \langle \sigma, TR, IC, S_0 \rangle$ we perform further translation as follows. For every functional symbol f of the sort $S_1 \times \ldots \times S_k \rightarrow S$ introduce corresponding predicate symbol val_f of the sort $S_1 \times \ldots \times S_k \times S \rightarrow Bool$. Assuming val_f models the graph of f translate the formula to that in the new, purely relational vocabulary. For all replaced functional symbols $f_1, \ldots, f_s$ add $Fun_{f_1,\ldots,f_s}$, the formulae expressing functionality of all $val_{f_1}, \ldots val_{f_s}$, to $[\![IC]\!]^{\sigma}_{integrity}$. The finiteness conditions imposed by the definition of monodic rules, ensures that $Fun_{f_1,\ldots,f_s}$ can be expressed without equality.

Given that $[\![TR]\!]_{frame}$ and $[\![TR]\!]_{rules}$ are in the relational vocabulary, further translation is performed as follows: replace all quantifiers over finite sorts with conjunctions and disjunctions over elements of the sort. Then, the first and second conditions of the definition of monodic rules ensure that, in the resulting formula, we have no more than one free variable in subformulae under the $\bigcirc$ operator in the *Persists* subformulae. It follows that the resulting formulae $[\![TR]\!]^*_{rules}$ and $[\![TR]\!]^*_{frame}$ are monodic. Further, the second and third conditions of the definition of monodic rules ensure that, in the translation of frame conditions, one can remove all variables in atomic formulae with equality in the Φ_f subformulae, except possibly one for the main functional symbol f. All possible remaining occurences of equality are of the form $\ldots[\ldots(x = a)\ldots \Rightarrow Persists(\ldots)]$.

To illustrate this point consider as an example a rule **if** true **then** $f(5) = 3$, where the constant 5 belongs to some infinite sort and f is a finitely valued functional symbol. Then, the corresponding frame condition would be $\Box\forall v, w((v \neq 5) \rightarrow \mathit{Persists}(f(v) = w))$. One can eliminate an equality in subformula $f(v) = w$, but not in $v \neq 5$ because of the infinite nature of the sort. That ensures the resulting formulae are in the packed monodic fragment of FOTL. If condition $(\mathbf{a})$ of the definition of monodic ASM programs holds then all equalities can be eliminated and $[\![\mathcal{M}]\!]$ will be in the monodic fragment without equality. Otherwise the condition $(\mathbf{b})$ holds and, together with the above observations, this ensures that $[\![\mathcal{M}]\!]$ is in the packed monodic fragment (with equality). $\Box$

6 Where are the Monodic ASMs?

The notion of monodic ASM presented in the previous section is fairly restricted. The area of its applicability still needs to be understood better. In this section we present two examples of realistic distributed algorithms (protocols) which can be specified and verified via approach we have presented.

6.1 FloodSet algorithm

We consider here a variant of the *FloodSet algorithm with alternative decision rule* (in terms of [29], p.105) designed for solution of the Consensus problem in the presence of crash (or fail-stop) failures.

The setting is as follows. There are n processes, each having an *input bit* and an *output bit*. The processes work synchronously, run the same algorithm and use *broadcast* for communication. Some processes may fail and, from that point onward, such processes do not send any further messages. Note, however, that the messages sent by a process *in the moment of failure* may be delivered to *an arbitrary subset* of the processes. Crucially, there is a bound, f, on the number of processes that may fail.

The goal of the algorithm is to eventually reach an agreement, i.e. to produce an output bit, which would be the same for all non-faulty processes. It is required also that if all processes have the same input bit, that bit should be produced as an output bit.

This protocol (adapted from [29][2]) is as follows.

- At the first round of computations, every process broadcasts its input bit.
- At every later round, a process broadcasts any value *the first time it sees it*.
- At every round the (tentative) output bit is set to the minimum value ever seen so far.

The correctness criteria for this protocol is that, eventually (actually, no later than in $f+2$ rounds) the output bits of all non-faulty processes will be the same.

[2] The difference being that, in [29], every process *knows* the bound f in advance and stops the execution of the protocol after $f+2$ rounds, producing the appropriate output bit. For generality, we consider the version where the processes do not know f in advance and produce a *tentative output bit* at every round.

Claim. The above FloodSet algorithm can be specified (naturally) as a monodic ASM $\mathcal{M}$. Its temporal translation $[\![\mathcal{M}]\!]$ and correctness condition are in the decidable two-variable monodic without equality fragment of FOTL under finite domain semantics.

We justify this claim in the next sections.

6.2 ASM Specification

Define ASM specification $\mathcal{M} = \langle \sigma, TR, IC, \Psi_0 \rangle$ of the above *FloodSet* algorithm as follows.

Formally, the states of our ASMs are many-sorted first-order structures, consisting of the sorts *Processes*, *Values* and *Bool* = $\{true, false\}$. For simplicity, we consider the particular case with *Values* = $\{0, 1\}$. The disjont parts σ_d, σ_s and σ_e of vocabulary σ consist of the following functions toghether with their associated sorts:

Dynamic, σ_d:

- $1stRound : Bool$
- $Normal : Processes \to Bool$
- $Faulty : Processes \to Bool.$
- $Output : Processes \to Values.$
- $Send : Processes \times Values \to Bool.$
- $Send\_failure : Processes \times Values \to Boolean.$
- $Received : Processes \times Values \to Bool$
- $Seen : Processes \times Values \to Bool.$

Static, σ_s:

- $Input : Processes \to Values$

External, σ_e:

- $Failure : Processes \to Bool.$
- $Deliver\_failure : Processes \times Processes \to Boolean.$

We put $Input$ function into the Static part, because once a process received some value as an input at the beginning (this is encoded at the initial state of ASM) it is going to keep it unchanged for the whole computation. On the other hand we consider *Failure* function as an external one, hereby modelling the failure of a process as an effect of oracle, which 'chooses' an arbitrary set of failed processes at every step. Another external function is $Deliver\_failure$ which determines the set of processes to which a message to be delivered if sent by a process in *Failure* state.

Given this signature, the ASM program (specification) for the FloodSet algorithm we are considering is given in Fig. 2. Note that, for readability, we replace occurrences of $P(t_1, \dots t_n) = true(false)$ and $Q(t_1, \dots t_n) := true(false)$ in the rules with $(\neg)P(t_1, \dots t_n)$ and $(\neg)Q(t_1, \dots t_n)$, respectively. Also for readability we present ASM in not a normal form.

TRANSITION RULES, TR:

if $1stRound$ **then** $\neg 1stRound$

if $Normal(x) \wedge Failure(x)$ **then** $Faulty(x) \wedge \neg Normal(x)$

if $1stRound \wedge Normal(x)$ **then** $Send(x, Input(x))$, $Seen(x, Input(x))$

if $1stRound \wedge Failure(x)$ **then** $Send\_Failure(x, Input(x))$, $Seen(x, Input(x))$

if $\neg 1stRound \wedge Normal(x) \wedge \neg Failure(x) \wedge Received(x,y) \wedge \neg Seen(x,y)$ **then** $Seen(x,y)$, $Send(x,y)$

if $\neg 1stRound \wedge Failure(x) \wedge Received(x,y) \wedge \neg Seen(x,y)$ **then** $Seen(x,y)$, $Send\_Failure(x,y)$

if $(\neg Received(x,y) \vee Seen(x,y))$ **then** $\neg Send(x,y), \neg Send\_Failure(x,y)$

if $Send(x,m)$ **then** $Received(y,m)$

if $Send\_Failure(x,m) \wedge Deliver\_Failure(x,y)$ **then** $Received(y,m)$

if $\neg Send(x,m) \wedge \neg(Send\_Failure(x,m) \wedge Deliver\_Failure(x,y))$ **then** $\neg Received(y,m)$

if $Faulty(x)$ **then** $\neg Send(x,y)$, $\neg Send\_failure(x,y)$

if $\neg Faulty(x) \wedge Seen(x,0)$ **then** $Output(x) := 0$

if $\neg Faulty(x) \wedge \neg Seen(x,0) \wedge Seen(x,1)$ **then** $Output(x) := 1$

INTEGRITY CONSTRAINTS, IC:

None

INITIAL CONDITIONS Ψ_0:

$1stRound$

$\forall x Normal(x)$

$\forall x \forall y \neg Seen(x,y)$

Fig. 2. ASM Specification of the FloodSet Algorithm

6.3 Temporal Translation of FloodSet ASM

In this section we demonstrate temporal tranlsation of the FloodSet ASM specification. The vocabulary of the (first stage of) FOTL translation consists of the binary and unary predicates instead of binary and unary functions of boolean type and propositional symbols instead 0-ary functions that occurred in the ASM specification. To indicate the close correspondence, we re-use the same names.

A (one-sorted) FOTL representation φ of above ASM program is a conjunction of the following formulae, divided into four groups as follows.

1. **INITIAL CONDITIONS:** (i.e. $[\![\Psi_0]\!]^{\sigma}_{initial}$)
   $\Box(\mathbf{start} \Rightarrow 1stRound)$
   $\Box(\mathbf{start} \Rightarrow \forall x\, Normal(x))$
   $\Box(\mathbf{start} \Rightarrow \forall x \forall y\, \neg Seen(x,y))$
2. **STATIC FUNCTIONS:** (i.e. $[\![\sigma]\!]_{static}$)
   $\Box(\forall x,y\, (Input(x) = y) \rightarrow \bigcirc(Input(x) = y))$
3. **SORT CONSTRAINTS**
   $\Box(\forall x(Output(x) = 0 \vee Output(x) = 1))$
   $\Box(\forall x(Input(x) = 0 \vee Input(x) = 1))$
   $\Box(\forall x \forall y(Send(x,y) \vee Received(x,y) \vee Seen(x,y)) \rightarrow (y = 0 \vee y = 1))$
   ...

Comment: We omit constraints for the sort *Processes* because it is *only* infinite sort here and such ommission would not affect verification.

1. **RULES:** (i.e. $[\![TR]\!]^{\sigma}_{rules}$)
   $\Box(1stRound \rightarrow \bigcirc\neg 1stRound)$
   $\Box(\forall x(Normal(x) \wedge Failure(x) \rightarrow \bigcirc(Faulty(x) \wedge \neg Normal(x))))$
   $\Box(\forall x(1stRound \wedge Normal(x) \rightarrow \bigcirc(Send(x,Input(x)) \wedge Seen(x,Input(x)))))$
   $\Box(\forall x(1stRound \wedge Failure(x) \rightarrow \bigcirc(Send\_Failure(x,Input(x)) \wedge Seen(x,Input(x)))))$
   $\Box(\forall x \forall y(\neg 1stRound \wedge Normal(x) \wedge \neg Failure(x) \wedge Received(x,y) \wedge \neg Seen(x,y) \rightarrow \bigcirc Seen(x,y) \wedge Send(x,y)))$
   $\Box(\forall x \forall y(\neg 1stRound \wedge Failure(x) \wedge Received(x,y) \wedge \neg Seen(x,y) \rightarrow \bigcirc(Seen(x,y) \wedge Send\_Failure(x,y))))$
   $\Box(\forall x \forall y((\neg Received(x,y) \vee Seen(x,y)) \rightarrow \bigcirc(\neg Send(x,y) \wedge \neg Send\_Failure(x,y))))$
   $\Box(\forall x \forall y \forall m Send(x,m) \rightarrow \bigcirc Received(y,m))$
   $\Box(\forall x \forall y \forall m(Send\_Failure(x,m) \wedge Deliver\_Failure(x,y) \rightarrow \bigcirc Received(y,m))$
   $\Box(\forall x \forall y \forall m(\neg Send(x,m) \wedge \neg(Send\_Failure(x,m) \wedge Deliver\_Failure(x,y)) \rightarrow \bigcirc\neg Received(y,m)))$
   $\Box\forall x \forall y(Faulty(x) \rightarrow \bigcirc\neg Send(x,y) \wedge \neg Send\_failure(x,y))$
   $\Box(\forall x(\neg Faulty(x) \wedge Seen(x,0) \rightarrow \bigcirc Output(x) = 0))$
   $\Box(\forall x(\neg Faulty(x) \wedge \neg Seen(x,0) \wedge Seen(x,1) \rightarrow \bigcirc Output(x) = 1))$

Comment: The above formulae from the **Rules** part are obtained by obvious simplification of those from $[\![TR]\!]^{\sigma}_{rules}$. For example, official translation of the rule '**if** $Normal(x) \wedge Failure(x)$ **then** $Faulty(x); \neg Normal(x)$' is $\Box\forall x \forall v[Normal(x) \wedge Failure(x) \wedge (v = x) \rightarrow \bigcirc Faulty(v) \wedge \neg Normal(x)]$, which becomes $\Box(\forall x(Normal(x) \wedge Failure(x) \rightarrow \bigcirc Faulty(x)) \wedge \neg Normal(x))$ after simplification. Simplification of the formulae including the *Input* function relies also on Persistence of *Input* (cf. **Static functions** part).

4 **FRAME CONDITIONS:** (i.e. $[\![TR]\!]^{\sigma}_{frame}$)

$\Box\forall z(\neg 1stRound \to Persists(1stRound = z))$
$\Box\forall x,z(\neg(Normal(x)\wedge Failure(x)) \to Persists(Faulty(x) = z))$
$\Box\forall x,z(\neg(Normal(x)\wedge Failure(x)) \to Persists(Normal(x) = z))$
$\Box\forall x,z((Faulty(x)\vee\neg Seen(x,0))\wedge(Faulty(x)\vee Seen(x,0)\vee\neg Seen(x,1)) \to Persists(Output(x) = z)$
$\Box\forall x,y,z((y \neq Input(x)\vee\neg 1stRound\vee\neg Normal(x))\wedge(1stRound\vee\neg Normal(x)\vee Failure(x)\vee\neg Received(x,y)\vee Seen(x,y))\wedge(Received(x,y)\wedge\neg Seen(x,y))\wedge(\neg Faulty(x))) \to Persists(Send(x,y) = z)$

$\Box\forall x,y,z((y \neq Input(x)\vee\neg 1stRound\vee\neg Failure(x))\wedge(1stRound\vee\neg Failure(x)\vee\neg Received(x,y)\vee Seen(x,y))\wedge(Received(x,y)\wedge\neg Seen(x,y))\wedge(\neg Faulty(x)) \to Persists(Send\_Failure(x,y) = z))$

$\Box\forall x,y,m,z((\neg Send(x,m)\wedge\neg(Send\_Failure(x,m)\wedge Deliver\_Failure(x,y))\wedge\neg(\neg Send(x,m)\wedge\neg(Send\_Failure(x,m)\wedge Deliver\_Failure(x,y)))) \to \bigcirc Persists(Received(y,m) = z))$

$\Box\forall x,y,z((y \neq Input(x)\vee\neg 1stRound\neg Normal(x))\wedge(y \neq Input(x)\vee\neg 1stRound\vee\neg Failure(x))\wedge(1stRound\vee\neg Normal(x)\vee Failure(x)\vee\neg Received(x,y)\vee Seen(x,y))\wedge(1stRound\vee\neg Failure(x)\vee\neg Received(x,y)\vee Seen(x,y))) \to Persists(Seen(x,y) = z)$

Comment: The last four frame conditions look complicated. They may be simplified further. For example, left-hand side of implication in the frame condition for *Received* function is inconsistent. That means this frame condition is a tautology and may be ommited from specification. The frame condition for *Seen* can be simplified to

$$\Box\forall x,y,z((y \neq Input(x)\vee\neg 1stRound\vee(\neg Normal(x)\wedge\neg Failure(x)))\wedge$$

$$(1stRound\vee(\neg Normal(x)\wedge\neg Failure(x))\vee\neg Received(x,y)\vee Seen(x,y))) \to Persists(Seen(x,y) = z)$$

Reducing to monodic specification: The above temporal specification is not monodic as it is. To make it monodic we replace all quantifiers over finite sorts by explicite conjunctions and disjunctions. We inroduce also the new predicate name *val_Output* and replace occurences of *Output* accordingly.

Some examples of resulting formulae are in order.

From Rules part:

$$\Box(\forall x((\neg Received(x,0)\vee Seen(x,0)) \to \bigcirc(\neg Send(x,0)\wedge\neg Send\_Failure(x,0))))$$

$$\wedge$$

$$\Box(\forall x((\neg Received(x,1)\vee Seen(x,1)) \to \bigcirc(\neg Send(x,1)\wedge\neg Send\_Failure(x,1))))$$

and

$$\Box(\forall x(\neg Faulty(x)\wedge\neg Seen(x,0)\wedge Seen(x,1) \to \bigcirc val\_Output(x,1)))$$

From Frames part:

$$\Box\forall x,z(\neg(Normal(x)\wedge Failure(x))\rightarrow Persists(Faulty(x))$$

$$\wedge$$

$$\Box\forall x,z(\neg(Normal(x)\wedge Failure(x))\rightarrow Persists(\neg Faulty(x))$$

Correctness Conditions: The correctness condition for above protocol is expressed by the FOTL formula χ:

$$\Diamond\begin{bmatrix}(\forall x\neg Faulty(x)\rightarrow val\_Output(x,0))\\ \vee\\ (\forall x\neg Faulty(x)\rightarrow val\_Output(x,1))\end{bmatrix}$$

Note, however, that the correctness of the protocol holds only with additional assumption, that is the number of failed processes is bounded by finite constant. This assumption will hold if we consider finite domain semantics for the temporal specification and correctness conditions. Thus one can reduce verification of the FloodSet algorithm to the checking of validity of the first-order temporal formula which belong to decidable two-variable monodic fragment of FOTL over finite domains.

6.4 Cache coherence protocols

Another class of protocols which can be specified via monodic ASMs, and for which correctness can be reduced to the validity of monodic temporal formulae, is the class of *cache coherence protocols* [22]. These are designed to provide data consistency betwen caches of different processors in shared-memory multiprocessor systems, i.e. to ensure that any copies of the same memory block in the caches of different processors are identical. Abstracting from the low-level implementation details of *read*, *write* and *syncronisation* primitives, cache coherence protocols [22, 10] can be seen as describing the families of *identical* finite state machines (each being cache controller of a processor) together with a simple form of communication: when one machine makes a transition (an *action*) a, the the other machines are required to do complementary transition (a *re-action*) $\bar{a}$. Correctness can then be expressed in terms of co-occurences some states in which, for example, the processors are allowed to change the content of their caches.

In [14] we have shown that theorem proving in monodic temporal fragments can be used for the verification of cache coherence protocols for an arbitrary number of processors. Here, we illustrate the point that *natural* ASM specifications of finite state based models of cache coherence protocols can be translated into temporal monodic specifications by the above translation procedure. As an example we consider "Synapse N+1" protocol which is particularly simple, consisting of only 3 states, 2 actions (transitions), *Read* and *Write* and two re-actions (complementary transitions) $\overline{Read}$ and $\overline{Write}$.

6.5 ASM specification of Synapse N+1 protocol

In the ASM specification below we assume

- the elements of the domain represent cache controllers of different processors;
- states of cache controllers are represented by unary predicates holding or not on the corresponding elements of the domain, e.g. $P(x)$ means "automaton x is in state P";
- use of only two (external) predicates $Read$ and $Write$ for actions, while reactions $\overline{Read}$ and $\overline{Write}$ are expressed in terms of these two.

We define and ASM specification $\mathcal{M} = \langle \sigma, TR, IC, \Psi_0 \rangle$ of the *Synapse N+1* algorithm as follows. The states are two-sorted first-order structures, consisting of the sorts *Processors* and *Bool* = $\{true, false\}$. The disjont parts σ_d, σ_s and σ_e of vocabulary σ consist of the following functions together with their associated sorts:

Dynamic, σ_d:
- $Initial : Bool$
- $Valid : Processors \to Bool$
- $Invalid : Processors \to Bool.$
- $Dirty : Processors \to Bool.$

Static, σ_s:
None

External, σ_e:
- $Read : Processors \to Bool.$
- $Write : Processors \to Bool$

Correctness Conditions:

These are defined as

χ_1: $\Box\, \forall x \forall y\, (\neg D(x) \vee \neg V(y))$
χ_2: $\Box\, \forall x \forall y\, (D(x) \wedge D(y) \to x = y)$

It is easy to check that both ASM specification and correctness conditions are monodic. We restrict ourselves with some comments on the resulting temporal translation. The components $[\![\Psi_0]\!]^{\sigma}_{initial}$, $[\![\sigma]\!]_{static}$, $[\![\sigma]\!]_{sort}$ and $[\![\sigma]\!]_{integrity}$ of the translation are straighforward. The rules part $[\![TR]\!]^{\sigma}_{rules}$ after easy simplification also takes an expected form, including e.g.

$$\Box(\forall x\, (Valid(x) \wedge \neg Read(x) \wedge \exists y Read(y)) \to \bigcirc\, Valid(x))$$

The frame part $[\![TR]\!]^{\sigma}_{frame}$ after simplification will contain all frame conditions, including e.g.

$$\Box(\neg(Invalid(x) \wedge Read(x)) \wedge \neg(Valid(x) \wedge Read(x)) \wedge \neg(Valid(x) \wedge \neg Read(x) \wedge \exists y Read(y)) \to$$

$$Persists(\neg Valid(x))$$

TRANSITION RULES, TR:

if $Initial$ **then** $\neg Initial$

if $Invalid(x) \wedge Read(x)$ **then** $Valid(x)$

if $Invalid(x) \wedge \neg Read(x) \wedge \exists y Read(y)$ **then** $Invalid(x)$

if $Invalid(x) \wedge Write(x)$ **then** $Dirty(x)$

if $Invalid(x) \wedge \neg Write(x) \wedge \exists y Write(y)$ **then** $Invalid(x)$

if $Valid(x) \wedge Read(x)$ **then** $Valid(x)$

if $Valid(x) \wedge \neg Read(x) \wedge \exists y Read(y)$ **then** $Valid(x)$

if $Valid(x) \wedge Write(x)$ **then** $Dirty(x)$

if $Valid(x) \wedge \neg Write(x) \wedge \exists y Write(y)$ **then** $Invalid(x)$

if $Dirty(x) \wedge Read(x)$ **then** $Dirty(x)$

if $Dirty \wedge \neg Read(x) \wedge \exists y Read(y)$ **then** $Invalid(x)$

if $Dirty(x) \wedge \neg Write(x)$ **then** $Dirty(x)$

if $Dirty(x) \wedge \neg Write(x) \wedge \exists y Write(y)$ **then** $Invalid(x)$

INTEGRITY CONSTRAINTS, IC:

$\exists\,!\, x(Read(x) \mid Write(x))$

$\forall x\ (V(x) \mid I(x) \mid D(x))$

INITIAL CONDITIONS Ψ_0:

$\forall x\, Invalid(x)$

Fig. 3. ASM Specification of the Synapse N+1 protocol

In general, since the obtained temporal formulae are monodic, we must consider the effect of the use of the equality predicate. It is known that even severely restricted classes of monodic formulae with equality may have highly undecidable validity checking problem [8]. However, as is shown in [14] the proof methods for the fragment without equality [4] may be used for the proof of the above correctness condition χ_1 (and such like). As to the χ_2, the equality may be abstracted away, by renaming the entire formula by a propositional letter and adding a corresponding $\Box$-closed equivalence to the specification. After that the same procedure proves the validity.

Note 5. Unlike the case of the FloodSet protocol, the Frame Conditions here are, in principle, redundant as they are implied by the rest of temporal translation. The reason for this is that an ASM specification reflecting a finite state model describes *all* transitions explicitly, including those which do not change states. On the other hand, the frame conditions may turn out to be useful, if used as an additional lemma, during the proof of correctness. We leave investigation into the optimization of our temporal translations in general to future.

7 Related work

We have learned from one of the referee of the conference version of this paper that in [32] Antje Nowack has suggested to use monodic fragments of FOTL to verify ASMs.

In particular, he proposed the definition of *guarded* ASM and has shown that verification of the properties of such ASMs expressed in guarded monodic fragment of FOTL is decidable. A reduction to finite satisfiability problem of guarded monodic fragment is used.

It is turned out that our work (done independently) is very much in spirit of [32]. We formulated restrictions on ASM which ensure that temporal translation falls into monodic fragment, which is, in general, undecidable, but finitely axiomatizable and hence semi-decidable. Technically, our definition of monodic ASM is less restrictive than that of guarded ASMs. On the other hand we can only guarantee existence of semi-decision procedure for monodic ASM (and suitable correctness conditions) as opposite to decision procedure for guarded ASMs (and suitable correctness conditions). However, checking whether the result of our temporal translation is in a known decidable fragment of FOTL is straightforward, so one can apply a decision procedure if it is the case. If the translation is outside known decidable fragment one can apply incomplete methods fo verification. We have illustrated both these cases with the examples.

8 Conclusion

We have shown how to faithfully translate ASM specifications into the first-order temporal logic (FOTL) and have defined restrictions on ASM specifications (programs) which ensure that temporal translation is in a 'good' (monodic) fragment of FOTL. This allows us to use temporal translations for (semi-)automatic verifications of restricted ASM specifications either by decision procedures [7], or by theorem proving for restricted fragments of FOTL [4]. Design of effective automatic algorithms where possible, is an important future development, as is an in-depth analysis of the applicability of monodic ASMs.

Acknowledgements. The authors acknowledge partial support from EPSRC (through grant GR/M46631) for the work reported in this paper.

References

[1] Abstract State Machines: A Formal Method for Specification and Verification. [`http://www.eecs.umich.edu/gasm/`].

[2] A. Blass and Y. Gurevich. Abstract State Machines Capture Parallel Algorithms. *ACM Transactions on Computational Logic*, 2003. (To appear). An early version appeared under different titles as Microsoft Research Technical Reports MSR-TR-99-09 and MSR-TR-99-65, and in Bulletin of EATCS 67 (February 1999), 93-124.

[3] E. Börger. High Level System Design and Analysis using Abstract State Machines. In *Current Trends in Applied Formal Methods (FM-Trends 98)*, number 1641 in LNCS, pages 1–43. Springer-Verlag, 1999.

[4] J. Brotherston, A. Degtyarev, M. Fisher, and A. Lisitsa. Implementing Invariant Search via Temporal Resolution. In *Proc. LPAR-2002*. Springer Verlag, 2002. LNCS, to appear.

[5] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Dec. 1999.

[6] A. Degtyarev and M. Fisher. Towards First-Order Temporal Resolution. In *Proc. KI-2001*, pages 18–32. Springer Verlag, 2001. Volume 2174 of *LNAI*.

[7] A. Degtyarev, M. Fisher, and B. Konev. Monodic temporal resolution. Accepted to CADE'03. Available as Technical report ULCS-03-001 from `http://www.csc.liv.ac.uk/research/techreports`, 2003.

[8] A. Degtyarev, M. Fisher, and A. Lisitsa. Equality and Monodic First-Order Temporal Logic. *Studia Logica*, (72) 2002, PP 147–156.

[9] G. Del Castillo and K. Winter. Model Checking Support for the ASM High-Level Language. In *Proc. TACAS*, volume 1785 of *LNCS*, pages 331–346. Springer-Verlag, 2000.

[10] G. Delzanno, Constraint-based verification in Parametrized Cache Coherence Protocols, to appear in Formal Methods in System Design

[11] E. A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.

[12] M. Fisher. A normal form for temporal logics and its applications in theorem proving and execution. *Journal of Logic and Computation*, 7(4):429–456, 1997.

[13] M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, Jan. 2001.

[14] M. Fisher, A. Lisitsa. Deductive Verification of Cache Coherence Protocols. In Proceedings of the 3rd Workshop on Automated Verification of Critical Systems AVoCS'03, Southampton, UK, April 2003, pp 177–186

[15] M. Fisher, A. Lisitsa Temporal verification of monodic abstract state machines Technical Report ULCS-03-011, University of Liverpool Department of Computer Science, April 2003, avaialable at `http://www.csc.liv.ac.uk/research/techreports`

[16] A. Gargantini and E. Riccobene. Encoding Abstract State Machines in PVS. In *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 303–322. Springer-Verlag, 2000.

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

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

[19] Y. Gurevich. Logician in the land of OS: Abstract State Machines at Microsoft. In *Proc. 16th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 129–136. IEEE Computer Society, 2001.

[20] U. Glässer and Y. Gurevich and M. Veanes. Universal Plug and Play Models. Technical Report MSR-TR-2001-59, Microsoft Research, June 2001.

[21] Y. Gurevich and W. Schulte and M. Veanes. Toward Industrial Strength Abstract State Machines. Technical Report MSR-TR-2001-98, Microsoft Research, October 2001.

[22] J. Handy. The Cache memory Book. Academic Press, 1993.

[23] I. Hodkinson Monodic packed fragment with equality is decidable. *Studia Logica*, (72) 2002, pp 185–197.

[24] I. Hodkinson, F. Wolter, and M. Zakharyashev. Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic*, 2000.

[25] U. Hustadt and B. Konev. TRP 2.0: A temporal resolution prover. Accepted to CADE'03.

[26] U. Hustadt and R. A. Schmidt. Formulae which highlight differences between temporal logic and dynamic logic provers. In *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, Technical Report DII 14/01, pages 68–76. Dipartimento di Ingegneria dell'Informazione, Unversitá degli Studi di Siena, 2001.

[27] U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In *Proc. KR'2002*, pages 533–544. Morgan Kaufmann, 2002.

[28] M. Huth and M. Ryan. *Logic in Computer Science*. Cambridge University Press, 2000.

[29] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[30] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.

[31] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[32] A. Nowack. Deciding the Verification Problem fro Abstract State Machines. In Proceedings of ASM 2003, *LNCS*, 2589: 341–371, 2003.

[33] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.

[34] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λ*clam*. *LNCS*, 1421:129–??, 1998.

[35] Z. Manna and the STeP group. Step: Deductive–algorithmic verification of reactive and real-time systems. In *International Conference on Computer Aided Verification (CAV)*, volume 1102 of *LNCS*. Springer-Verlag, 1996.

[36] A. P. Sistla and E. M. Clarke. Complexity of propositional linear temporal logics. *ACM Journal*, 32(3):733–749, July 1985.

[37] M. Spielmann. Model Checking Abstract State Machines and Beyond. In *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 323–340. Springer-Verlag, 2000.

[38] K. Winter. Towards a Methodology for Model Checking ASM: Lessons Learned from the FLASH Case Study. In *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 341–360. Springer-Verlag, 2000.

[39] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[40] F. Wolter and M. Zakharyaschev. Axiomatizing the monodic fragment of first-order temporal logic. *Annals of Pure and Applied logic*, 118:133–145, 2002.