Towards SMT Model Checking of Array-based Systems

Silvio Ghilardi[1], Enrica Nicolini[2], Silvio Ranise[1,2], and Daniele Zucchelli[1]

[1] Dipartimento di Informatica, Università degli Studi di Milano (Italia)
[2] LORIA & INRIA-Lorraine, Nancy (France)

Abstract. We introduce the notion of array-based system as a suitable abstraction of infinite state systems such as broadcast protocols or sorting programs. By using a class of quantified-first order formulae to symbolically represent array-based systems, we propose methods to check safety (invariance) and liveness (recurrence) properties on top of Satisfiability Modulo Theories solvers. We find hypotheses under which the verification procedures for such properties can be fully mechanized.

1 Introduction

Model checking of infinite-state systems manipulating arrays – e.g., broadcast protocols, lossy channel systems, or sorting programs – is a hot topic in verification. The key problem is to verify the correctness of such systems regardless of the number of elements (processes, data, or integers) stored in the array, called *uniform verification problem*. In this paper, we propose *array-based systems* as a suitable abstraction of broadcast protocols, lossy channel systems, or, more in general, programs manipulating arrays (Section 3). The notion of array-based system is parametric with respect to a theory of indexes (which, for parametrized systems, specifies the topology of processes) and a theory of elements (which, again for parametrized systems, specifies the data manipulated by the system). Then (Section 3.1), we show how states and transitions of a large class of array-based systems can be symbolically represented by a class of *quantified* first-order formulae whose satisfiability problem is decidable under reasonable hypotheses on the theories of indexes and elements (Section 4). We also sketch how to extend the *lazy Satisfiability Modulo Theories* (SMT) techniques [16] by a suitable *instantiation strategy* to handle universally quantified variables (over indexes) so as to implement the satisfiability procedure for the class of quantified formulae under consideration (Figure 2). The capability to handle a (limited form) of universal quantification is crucial to reduce entailment between formulae representing states to satisfiability of a formula in the class of quantified formulae previously defined. This observation together with closure under pre-image computation of the kind of formulae representing unsafe states allows us to implement a *backward* reachability procedure (see, e.g., [12], or also [15] for a declarative approach) for checking safety properties of array-based systems (Section 5). In general, the procedure may not terminate;

but, under some additional assumptions on the theory of elements, we are able to prove termination by revisiting the notion of configuration (see, e.g., [1]) in first-order model-theory (Section 5.2). Finally (Section 6), we show the flexibility of our approach by studying the problem of checking the class of liveness properties called *recurrences* [13]. We devise deductive methods to check such properties based on the synthesis of so-called progress conditions at quantifier free level and we show how to fully mechanize our technique under the same hypotheses for the termination of the backward reachability procedure.

For space reasons, we did not include proofs; for the same reason, although our framework covers all examples discussed in [2,3] (and more), only one running example is discussed. For both proofs and examples, the reader is referred to the Technical Report [10].

2 Formal Preliminaries

We assume the usual first-order syntactic notions of signature, term, formula, quantifier free formula, and so on; equality is always included in our signatures. If $\underline{x}$ is a finite set of variables and Σ is a signature, by a $\Sigma(\underline{x})$-term, -formula, etc. we mean a term, formula, etc. in which at most the $\underline{x}$ occur free (notations like $t(\underline{x})$, $\varphi(\underline{x})$ emphasize the fact that the term t or the formula φ in in fact a $\Sigma(\underline{x})$-term or a $\Sigma(\underline{x})$-formula, respectively). The notions of interpretation, satisfiability, validity, and logical consequence are also the standard ones; when we speak about satisfiability (resp. validity) of a formula containing free variables, we mean satisfiability of its existential (resp. universal) closure. If $\mathcal{M} = (M, \mathcal{I})$ is a Σ-structure, a *Σ-substructure* of $\mathcal{M}$ is a Σ-structure having as domain a subset of M which is closed under the operations of Σ (in a Σ-substructure, moreover, the interpretation of the symbols of Σ is given by restriction). The Σ-structure *generated by a subset* X of M (which is not assumed now to be closed under the Σ-operations) is the smallest Σ-substructure of $\mathcal{M}$ whose domain contains X and, if this Σ-substructure coincides with the whole $\mathcal{M}$, we say that X *generates* $\mathcal{M}$. A *Σ-embedding* (or, simply, an embedding) between two Σ-structures $\mathcal{M} = (M, \mathcal{I})$ and $\mathcal{N} = (N, \mathcal{J})$ is any mapping $\mu : M \longrightarrow N$ among the corresponding support sets which is an isomorphism between $\mathcal{M}$ and the Σ-substructure of $\mathcal{N}$ whose underlying domain is the image of μ (thus, in particular, for μ to be an embedding, the image of μ must be closed under the Σ-operations). A class $\mathcal{C}$ of structures is *closed under substructures* iff whenever $\mathcal{M} \in \mathcal{C}$ and $\mathcal{N}$ is (isomorphic to) a substructure of $\mathcal{M}$, then $\mathcal{N} \in \mathcal{C}$.

Contrary to previous papers of ours, we prefer to have here a more liberal notion of a theory, so we identify a *theory* T with a pair $(\Sigma, \mathcal{C})$, where Σ is a signature and $\mathcal{C}$ is a class of Σ-structures (the structures in $\mathcal{C}$ are called the *models* of T). The notion of *T-satisfiability* of φ means the satisfiability of φ in a Σ-structure from $\mathcal{C}$; similarly, *T-validity* of a sentence φ (noted $T \models \varphi$) means the truth of φ in all $\mathcal{M} \in \mathcal{C}$. The *Satisfiability Modulo Theory* T, SMT(T), problem amounts to establish the T-satisfiability of an arbitrary first-order formula (hence possibly containing quantifiers), w.r.t. some background theory T.

A *theory solver* for the theory T (T-solver) is any procedure capable of establishing whether any given finite conjunction of literals is T-satisfiable or not. The so-called lazy approach to solve SMT(T) problems for quantifier-free formulae consists of integrating a Boolean enumerator (usually based on a refinement of the DPLL algorithm) with a T-solver (see [14,16] for an overview). Hence, by assuming the existence of a T-solver, it is always possible to build a (lazy SMT) solver capable of checking the T-satisfiability of arbitrary Boolean combinations of atoms in T (or, equivalently, quantifier-free formulae). For efficiency, a T-solver is required to provide a more refined interface, such as returning a subset of a T-unsatisfiable input set of literals which is still T-unsatisfiable, called conflict set (again see [16] for details).

We say that T admits *quantifier elimination* iff for every formula $\varphi(\underline{x})$ one can compute a quantifier-free formula $\varphi'(\underline{x})$ which is T-equivalent to it (i.e. such that $T \models \forall \underline{x}(\varphi(\underline{x}) \leftrightarrow \varphi'(\underline{x}))$). Linear Arithmetics, Real Arithmetics, acyclic lists, and enumerated datatype theories (see below) admit elimination of quantifiers.

A theory $T = (\Sigma, \mathcal{C})$ is said to be *locally finite* iff Σ is finite and, for every finite set of variables $\underline{x}$, there are finitely many $\Sigma(\underline{x})$-terms $t_1, \ldots, t_{k_{\underline{x}}}$ such that for every further $\Sigma(\underline{x})$-term u, we have that $T \models u = t_i$ (for some $i \in \{1, \ldots, k_{\underline{x}}\}$). The terms $t_1, \ldots, t_{k_{\underline{x}}}$ are called $\Sigma(\underline{x})$-representative terms; if they are effectively computable from $\underline{x}$ (and t_i is computable from u), then T is said to be *effectively locally finite* (in the following, when we say 'locally finite', we in fact always mean 'effectively locally finite'). If Σ is finite and does not contain any function symbol (i.e. Σ is a purely relational signature), then any Σ-theory is effectively locally finite; other effectively locally finite theories are Boolean algebras and Linear Arithmetic modulo a fixed integer.

Let Σ be a finite signature; an *enumerated datatype theory* in Σ is a theory whose class of models contains only a single finite Σ-structure $\mathcal{M} = (M, \mathcal{I})$; we require $\mathcal{M}$ to have the additional property that for every $m \in M$ there is a constant $c \in \Sigma$ such that $c^{\mathcal{I}} = m$. It is easy to see that an enumerated datatype theory admits quantifier elimination (since $\exists x\, \varphi(x)$ is T-equivalent to $\varphi(c_1) \vee \cdots \vee \varphi(c_n)$, where $c_1, ..., c_n$ are interpreted as the finitely many elements of the enumerated datatype) and is effectively locally finite.

In the following, it is more natural to adopt a many-sorted language. All notions introduced above (such as term, formula, structure, and satisfiability) can be easily adapted to many-sorted logic, see e.g. Chapter XX of [8].

3 Array-based Systems and their Symbolic Representation

We develop a framework to state and solve model checking problems for safety and liveness of a particular class of infinite state systems by using deductive (more precisely, SMT) techniques. We focus on the class of systems whose state can be described by a finite collections of arrays, which we call *array-based* systems. As an example, consider parameterized systems, i.e. systems consisting of an arbitrary number of identical finite-state processes organized in a linear

array: a state a of such a parameterized system can be seen as a state of an array-based system (in the formal sense of Definitions 3.2 and 3.3 below), where the indexes of the domain of the function assigned to the state variable a are the identifiers of the processes and the elements of a are the data describing one of the finitely many states of each process.

Array-based Systems. To develop our formal model, we introduce three theories. We use two mono-sorted theories $T_I = (\Sigma_I, \mathcal{C}_I)$ (of indexes) and $T_E = (\Sigma_E, \mathcal{C}_E)$ (of elements), whose only sort symbol are called `INDEX` and `ELEM`, respectively. T_I specifies the 'topology' of the processes, e.g., in the case of parameterized systems informally discussed above, Σ_I contains only the equality symbol '=' and $\mathcal{C}_I$ is the class of all finite sets. Other interesting examples of T_I can be obtained by taking Σ_I to contain a binary predicate symbol R (besides =) and $\mathcal{C}_I$ to be the class of structures where R is interpreted as a total order, a graph, a forest, etc. For parameterized systems with finite-state processes, T_E can be the theory of an enumerated datatype. For the larger class of parameterized systems (e.g., the one considered in [2,3]) admitting integer (or real) variables local to each process, T_E can be the theory whose class of models consists of a single structure (like real numbers under addition and/or ordering). Notice that in concrete applications, T_E has a single model (e.g. an enumerate datatype, or the structure of real/natural numbers under suitable operations and relations), whereas T_I has many models (e.g. it has as models all sets, all finite sets, all graphs, all finite graphs, etc.). The technical hypotheses we shall need in Sections 4 and 5 on T_I and T_E are fixed in the following:

Definition 3.1. *An* index theory $T_I = (\Sigma_I, \mathcal{C}_I)$ *is a mono-sorted theory (let us call* `INDEX` *its sort) which is locally finite, closed under substructures and whose quantifier free fragment is decidable for T_I-satisfiability. An* element theory $T_E = (\Sigma_E, \mathcal{C}_E)$ *is a mono-sorted theory (let us call* `ELEM` *its sort) which admits quantifier elimination and whose quantifier free fragment is decidable for T_E-satisfiability.*

One may wonder how restrictive are the assumptions of the above definition: it turns out that they are very light (for instance, they do not rule out any of the examples considered in [2,3]). In fact, quantifier elimination holds for common datatype theories (integers, reals, enumerated datatypes, etc.) and the hypotheses on index theory are satisfied by most process 'topologies'.[3]

The third theory $A_I^E = (\Sigma, \mathcal{C})$ we need is obtained by combining an index theory T_I and an element theory T_E as follows. First, A_I^E has three sort symbols: `INDEX`, `ELEM`, and `ARRAY`; the signature Σ contains all the symbols in the disjoint union $\Sigma_I \cup \Sigma_E$ and a further binary function symbol *apply* of sort $\texttt{ARRAY} \times \texttt{INDEX} \longrightarrow \texttt{ELEM}$. (In the following, we abbreviate $apply(a, i)$ with $a[i]$, for a term of sort `ARRAY` and i term of sort `INDEX`.) Second, a three-sorted structure $\mathcal{M} = (\texttt{INDEX}^{\mathcal{M}}, \texttt{ELEM}^{\mathcal{M}}, \texttt{ARRAY}^{\mathcal{M}}, \mathcal{I})$ is in the class $\mathcal{C}$ iff $\texttt{ARRAY}^{\mathcal{M}}$ is the set

[3] They typically fail when processes are arranged in a ring (e.g. in the 'dining philosophers' example): rings require a unary function symbol to be formalized and the bijectivity constraint to be imposed is insufficient to make the theory locally finite.

of (total) functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}^{\mathcal{M}}$, the function symbol *apply* is interpreted as function application (i.e. as the standard reading operation for arrays), and $(\texttt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$, $(\texttt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ are models of T_I and T_E, respectively – here $\mathcal{I}_{|\Sigma_I}, \mathcal{I}_{|\Sigma_E}$ are the restriction of $\mathcal{I}$ to the symbols of Σ_I, Σ_E. (In the following, we use the notations $\mathcal{M}_I$ and $\mathcal{M}_E$ for $(\texttt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$ and $(\texttt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$, respectively.) If the model $\mathcal{M}$ of A_I^E is such that $\texttt{INDEX}^{\mathcal{M}}$ is a finite set, then $\mathcal{M}$ is called a *finite index model*.

For the remaining part of the paper, **we fix an index theory $T_I = (\Sigma_I, \mathcal{C}_I)$ and an element theory, $T_E = (\Sigma_E, \mathcal{C}_E)$** (we also let $A_I^E = (\Sigma, \mathcal{C})$ be the corresponding combined theory).

Once the theories constraining indexes and elements are fixed, we need the notions of state, initial state and state transition to complete our picture. In a symbolic setting, these are provided by the following definitions:

Definition 3.2. *An* array-based (transition) system (for (T_I, T_E)) *is a triple* $\mathcal{S} = (\underline{a}, I, \tau)$ *where:*

- $\underline{a}$ *is a tuple of variables of sort* `ARRAY` *(these are the* state variables*);*
- $I(\underline{a})$ *is a* $\Sigma(\underline{a})$*-formula (this is the* initial state formula*);*
- $\tau(\underline{a}, \underline{a}')$ *is a* $\Sigma(\underline{a}, \underline{a}')$*-formula – here* $\underline{a}'$ *is a renamed copy of the tuple* $\underline{a}$ *(this is the* transition *formula).*

Definition 3.3. *Let* $\mathcal{S} = (\underline{a}, I, \tau)$ *be an array-based transition system. Given a model* $\mathcal{M}$ *of the combined theory* A_I^E*, an* $\mathcal{M}$-state *(or, simply, a state) of* $\mathcal{S}$ *is an assignment mapping the state variables* $\underline{a}$ *to total functions* $\underline{s}$ *from the domain of* $\mathcal{M}_I$ *to the domain of* $\mathcal{M}_E$*. A* run *of the array-based system is a (possibly infinite) sequence* $\underline{s}_0, \underline{s}_1, \ldots$ *of states such that*[4] $\mathcal{M} \models I(\underline{s}_0)$ *and* $\mathcal{M} \models \tau(\underline{s}_k, \underline{s}_{k+1})$ *for* $k \geq 0$*.*

For simplicity, below, we assume that the tuple of array state variables $\underline{a}$ is a single variable a: all definitions and results of the paper can be easily generalized to the case of finitely many array variables and to the case of multi-sorted T_I, T_E (see [10] for a discussion on these topics).

3.1 Symbolic Representation of States and Transitions

The next step is to identify suitable syntactic restrictions for the formulae I, τ appearing in the definition of an array-based system. Preliminarily, we introduce some notational conventions that alleviate the burden of writing and understanding the various formulae for states and transitions: $d, e, \ldots$ range over variables of sort `ELEM`, $a, b \ldots$ over variables of sort `ARRAY`, $i, j, k, \ldots$ over variables of sort `INDEX`, and $\alpha, \beta, \ldots$ over variables of either sort `ELEM` or sort `INDEX`. An underlined variable name abbreviates a tuple of variables of unspecified (but finite) length; by subscripting with an integer an underlined variable name, we indicate the corresponding component of the tuple (e.g., $\underline{i}$ may abbreviate $i_1, \ldots, i_n$

[4] Notations like $\mathcal{M} \models I(\underline{s}_0)$ means that the formula $I(\underline{a})$ is true in $\mathcal{M}$ under the assignment mapping the $\underline{a}$'s to the $\underline{s}_0$'s.

and $\underline{i}_k$ indicates i_k, for $1 \leq k \leq n$). We also use $a[\underline{i}]$ to abbreviate the tuple of terms $a[i_1], \ldots, a[i_n]$. If $\underline{j}$ and $\underline{i}$ are tuples with the same length n, then $\underline{i} = \underline{j}$ abbreviates $\bigwedge_{k=1}^{n}(\underline{i}_k = \underline{j}_k)$. Possibly sub/super-scripted expressions of the forms $\phi(\underline{\alpha}), \psi(\underline{\alpha}), \ldots$ (with sub-/super-scripts) always denote **quantifier-free $(\Sigma_I \cup \Sigma_E)$-formulae in which at most the variables $\underline{\alpha}$ occur** (notice in particular that no array variable and no apply constructor $a[i]$ can occur here). Also, $\phi(\underline{\alpha}, \underline{t}/\underline{\beta})$ (or simply $\phi(\underline{\alpha}, \underline{t})$) abbreviates the substitution of the terms $\underline{t}$ for the variables $\underline{\beta}$ (now apply constructors may appear in t). Thus, for instance, when we write $\phi(\underline{i}, a[\underline{i}])$, we mean the formula obtained by the replacements $\underline{e} \mapsto a[\underline{i}]$ in the quantifier free formula $\phi(\underline{i}, \underline{e})$ (the latter contains at most the element variables $\underline{e}$ and at most the index variables $\underline{i}$).

We are now ready to introduce suitably restricted classes of formulae for representing states and transitions of array-based systems.

States. An **$\exists^I$-formula** is a formula of the form $\exists \underline{i}\, \phi(\underline{i}, a[\underline{i}])$: such a formula may be used to model sets of *unsafe* states of parameterized systems, such as violations of mutual exclusion:

$$\exists i\, \exists j\, (i \neq j \wedge a[i] = \texttt{use} \wedge a[j] = \texttt{use}), \tag{1}$$

where `use` is a constant symbol of sort `ELEM` in an enumerated datatype theory. A **$\forall^I$-formula** is a formula of the form $\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}])$: this is logically equivalent to the negation of an $\exists^I$-formula and may be used to model *initial* sets of states of parameterized systems, such as "all processes are in a given state":

$$\forall i\, (a[i] = \texttt{idle}),$$

where `idle` is a constant symbol of sort `ELEM` in an enumerated datatype theory.

Transitions. The intuition underlying the class of formulae representing transitions of array-based systems can be found by analyzing the structure of transitions of parameterized systems. In such systems, a transition has typically two components. The *local* component (see the formula ϕ_L in the Definition 3.4 below) specifies the transitions of a given fixed number of processes; the *global* component (see the formula ϕ_G in the Definition 3.4 below) specifies the transitions done by all the other processes in the array as a reaction to those taken by the processes involved in the local component. We are lead to the following:

Definition 3.4. *Consider formulae $\phi_L(\underline{i}, \underline{d}, \underline{d}')$ and $\phi_G(\underline{i}, \underline{d}, \underline{d}', j, e, e')$, where ϕ_G satisfies the following* seriality *requirement:*

$$A_I^E \models \forall \underline{i}\, \forall \underline{d}\, \forall \underline{d}'\, \forall j\, \forall e\, \exists e'\, \phi_G(\underline{i}, \underline{d}, \underline{d}', j, e, e'). \tag{2}$$

The **T-formula** *with local component ϕ_L and global component ϕ_G is the formula*

$$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}], a'[\underline{i}]) \wedge \mathit{Update}_G(\underline{i}, a, a')), \tag{3}$$

where we used the explicit definition (i.e. the abbreviation)

$$\mathit{Update}_G(\underline{i}, a, a') :\Leftrightarrow \forall j\, \phi_G(\underline{i}, a[\underline{i}], a'[\underline{i}], j, a[j], a'[j]). \tag{4}$$

Recall that, according to our conventions, the local component ϕ_L and the global component ϕ_G are quantifier-free $(\Sigma_I \cup \Sigma_E)$-formulae. In $\phi_G(\underline{i}, \underline{d}, \underline{d}', j, e, e')$ the variables $\underline{i}, \underline{d}, \underline{d}'$ are called *side* parameters (these variables are instantiated in (3)-(4) by $\underline{i}, a[\underline{i}], a'[\underline{i}]$, respectively) and the variables j, e, e' are called *proper* parameters (these variables are instantiated in (3)-(4) by $j, a[j], a'[j]$, respectively). The intuition underlying the formula $Update_G(\underline{i}, a, a')$ defined by (4) is that the array a' is obtained as *a global update of the array a according to* ϕ_G. In fact, in (4), the proper parameters of ϕ_G are instantiated as $j, a[j], a'[j]$, i.e. as the index to be updated, the old, and the new content of that index. Notice also that this updating is largely non-deterministic, because a T-formula like (3) does not univocally characterize the system evolution: this is revealed by the fact that ϕ_G is not a definable function, and by the fact that the side parameters $\underline{d}'$ that may occur in ϕ_G are instantiated as the updated values $a'[\underline{i}]$ (this circularity makes sense only if we view the T-formula (3) as expressing a *constraint* – not necessarily an assignment – for the updated array a').

In the remaining part of the paper, **we fix an array-based system $\mathcal{S} = (a, I, \tau)$, in which the initial formula I is a $\forall^I$-formula and the transition formula τ is a *disjunction of* T-formulae**.

Example 3.5. We present here the formalization into our framework of the *Simplified Bakery Algorithm* that can be found for instance in [3]. We take as T_I the pure equality theory in the signature $\Sigma_I = \{=\}$; to introduce T_E, we analyze which kind of local data we need. The data e appearing in an array of processes are records consisting of the following two fields: (i) the field $e.s$ represents the status ($e.s \in \{\texttt{idle}, \texttt{wait}, \texttt{use}, \texttt{crash}\}$); (ii) the field $e.t$ represents the ticket (tickets range in the real interval $[0, \infty)$, seen as a linear order with first element 0). Thus we need a two-sorted T_E and two array variables in the language (or, a single array variable and a three-sorted T_E, comprising a sort for the cartesian product): as pointed out above, such multi-sorted extensions of our framework are indeed straightforward. Models for T_E are the obvious ones: one sort is interpreted as an enumerated datatype and the other sort is interpreted as the positive real domain. The initial formula I is $\forall i\,(a[i].s = \texttt{idle})$. The transition $\tau(a, a')$ is the disjunction $\tau_1(a, a') \vee \tau_2(a, a') \vee \tau_3(a, a')$, where the T-formulae τ_n $(n \leq 3)$ have the standard format from Definition 3.4 – namely $\exists i\,(\phi_L^n(i, a[i], a'[i]) \wedge Update_G^n(i, a, a'))$ – and the local and the global components ϕ_L^n, ϕ_G^n are specified in Figure 1 below. When formalizing the component transitions τ_1, τ_2, we use the *approximation* trick (see [2,3]): transitions τ_1, τ_2, in order to fire, would require a universally quantified guard which cannot be expressed in our format. We circumvent the problem by imposing that all the processes that are counterexamples to the guard go into a 'crash' state: this trick augments the possible runs of the system by introducing 'spurious runs', however if a safety property holds for the augmented 'approximate' system, then it also holds for the original system (the same is true for the recurrence properties of Section 6). $\dashv$

Local components. The local components ϕ_L^n, for $n \leq 3$, are

- $\phi_L^1 \quad :\equiv \quad a[i].s = \texttt{idle} \wedge a'[i].s = \texttt{wait} \wedge a'[i].t > 0$ (the selected process goes from the state `idle` to `wait` and takes a nonzero ticket);
- $\phi_L^2 \quad :\equiv \quad a[i].s = \texttt{wait} \wedge a'[i] = \langle \texttt{use}, a[i].t \rangle$ (the selected process goes from the state `wait` to `use` and keeps the same ticket);
- $\phi_L^3 \quad :\equiv \quad a[i].s = \texttt{use} \wedge a'[i] = \langle \texttt{idle}, 0 \rangle$ (the selected process goes from the state `use` to `idle` and the ticket is reset to zero).

Global components. The global components $\phi_G^n(i, a[i], a'[i], j, a[j], a'[j])$, for $n \leq 3$, make case distinction as follows (the first case delegates the update to the local component):

- $\phi_G^1 \quad :\equiv \quad (j = i \wedge a'[j] = a'[i]) \vee (j \neq i \wedge a[j].t < a[i].t \wedge a'[j] = a[j]) \vee (j \neq i \wedge a[j].t \geq a[i].t \wedge a'[j] = \langle \texttt{crash}, a[j].t \rangle)$ (if the selected process i goes from `idle` to `wait`, then any other process j keeps the same state/ticket, unless it has a ticket bigger or equal to the ticket of i – in which case j crashes);
- $\phi_G^2 \quad :\equiv \quad (j = i \wedge a'[j] = a'[i]) \vee (j \neq i \wedge (a[j].t > a[i].t \vee a[j].t = 0) \wedge a'[j] = a[j]) \vee (j \neq i \wedge \neg(a[j].t > a[i].t \vee a[j].t = 0) \wedge a'[j] = \langle \texttt{crash}, a[j].t \rangle)$ (if the selected process i goes from `wait` to `use`, then any other process j keeps the same state/ticket, unless it has a nonzero ticket smaller than the ticket of i – in which case j crashes);
- $\phi_G^3 \quad :\equiv \quad (j = i \wedge a'[j] = a'[i]) \vee (j \neq i \wedge a'[j] = a[j])$ (if the selected process i goes from `use` to `idle`, then any other process keeps the same state/ticket).

Fig. 1. The Simplified Bakery Transition

4 Symbolic Representation and SMT solving

To make effective use of our framework, we need to be able to check whether certain requirements are met by a given array-based system. In other words, we need a powerful reasoning engine: it turns out that A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences introduced below is just what we need.[5]

Theorem 4.1. *The A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences, i.e. of sentences of the kind*

$$\exists a_1 \cdots \exists a_n \; \exists \underline{i} \; \forall \underline{j} \; \psi(\underline{i}, \underline{j}, a_1[\underline{i}], \ldots, a_n[\underline{i}], a_1[\underline{j}], \ldots, a_n[\underline{j}]), \tag{5}$$

is decidable.

We leave to [10] the detailed proof of the above theorem, here we focus on a discussion oriented to the employment of SMT-solvers in the decision procedure: Figure 2 depicts an SMT-based decision procedure for $\exists^{A,I}\forall^I$-sentences (in the simplified case in which only one variable of sort `ARRAY` occurs).

[5] Decidability of A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences plays in this paper a role similar to the Σ_1^0-decidability result employed in [4].

function A_I^E-check($\exists a\ \exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, a[\underline{i}], a[\underline{j}])$)
  $\underline{t} \longleftarrow$ compute-reps$_{T_I}(\underline{i})$; $\phi \longleftarrow \underline{t} = \underline{l}$; $Atoms \longleftarrow \mathsf{IE}(\underline{l})$
  for each substitution σ mapping the $\underline{j}$ into the $\underline{t}$'s **do**
    $\phi' \longleftarrow$ purify$(\psi(\underline{i}, \underline{j}\sigma, a[\underline{i}], a[\underline{j}\sigma]))$; $\phi \longleftarrow \phi \wedge \phi'$; $Atoms \longleftarrow Atoms \cup$ atoms(ϕ');
  end for each
  while Bool(ϕ) = sat **do**
    $\beta_I \wedge \beta_E \wedge \eta_I \longleftarrow$ pick-assignment$(Atoms, \phi)$
    $(\rho_I, \pi_I) \longleftarrow T_I$-solver$(\beta_I \wedge \eta_I)$
    $(\rho_E, \pi_E) \longleftarrow T_E$-solver$(\beta_E \wedge \bigwedge_{\underline{l}_{s_1} = \underline{l}_{s_2} \in \eta_I} (\underline{e}_{s_1} = \underline{e}_{s_2}))$
    if ρ_I = sat **and** ρ_E = sat **then return** sat
    if ρ_I = unsat **then** $\phi \longleftarrow \phi \wedge \neg\pi_I$
    if ρ_E = unsat **then** $\phi \longleftarrow \phi \wedge \neg\pi_E$
  end while
  return unsat
end

Fig. 2. The SMT-based decision procedure for $\exists^{A,I}\forall^I$-sentences

The set $\underline{t}$ of representative terms over $\underline{i}$ is computed by compute-reps$_{T_I}$ (to simplify the matter, we can freely assume $\underline{t} \supseteq \underline{i}$). This function is guaranteed to exist since T_I is effectively locally finite (for example, when Σ_I contains no function symbol, compute-reps$_{T_I}$ is the identity function). In order to purify formulae, we shall need below fresh index variables $\underline{l}$ abstracting the $\underline{t}$ and fresh element variables $\underline{e}$ abstracting the terms $a[\underline{l}]$: the conjunction of the 'defining equations' $\underline{t} = \underline{l}$ is stored as the formula ϕ, whereas the further defining equations $a[\underline{l}] = \underline{e}$ (not to be sent to the Boolean solver) will be taken into account inside the second loop body.

Then, the first loop is entered where we instantiate in all possible ways the universally quantified index variables $\underline{j}$ with the representative terms in the set $\underline{t}$. For every such substitution σ, the formula $\psi(\underline{i}, \underline{j}\sigma, a[\underline{i}], a[\underline{j}\sigma])$ is purified by replacing the terms $a[\underline{t}]$ with the element variables $\underline{e}$; after such purification, the resulting formula is added as a new conjunct to ϕ. Notice that this ϕ is a quantifier-free formula whose atoms are pure, i.e. they are either Σ_I- or Σ_E-atoms. The function $\mathsf{IE}(\underline{l})$ returns the set of all possible equalities among the index variables $\underline{l}$. Such equalities are added to the set of atoms occurring in ϕ as the T_I- and T_E-solvers need to take into account an equivalence relation over the index variables $\underline{l}$ so as to synchronize.

We can now enter the second (and main) loop: the Boolean solver, by invoking the interface function pick-assignment inside the loop, generates an assignment over the set of atoms occurring in ϕ and $\mathsf{IE}(\underline{l})$. The loop is exited when ϕ is checked unsatisfiable (test of the while). The T_I-solver checks for unsatisfiability the set β_I of Σ_I-literals in the Boolean assignment and the literals of the possible partition η_I on $\underline{l}$. Then, only the equalities $\bigwedge\{\underline{e}_{s_1} = \underline{e}_{s_2} \mid (\underline{l}_{s_1} = \underline{l}_{s_2}) \in \eta_I\}$ (and not also inequalities) among the purification variables $\underline{e}$ induced by the partition

η_I on $\underline{l}$ are passed to the T_E-solver together with the set β_E of Σ_E-literals in the Boolean assignment (this is to take into account the congruence of $a[\underline{l}] = \underline{e}$). If both solvers detect satisfiability, then the loop is exited and A_I^E-check also returns satisfiability. Otherwise (i.e. if at least one of the solvers returns unsat), the negation of the computed conflict set (namely, π_I or π_E) is conjoined to ϕ so as to refine the Boolean abstraction and the loop is resumed. If in the second loop, satisfiability is never detected for all considered Boolean assignments, then A_I^E-check returns unsatisfiability. The second loop can be seen as a refinement of the Delayed Theory Combination technique [5].

The critical point of the above procedure is the fact that the first loop may produce a very large ϕ: in fact, even in the most favourable case in which compute-reps$_{T_I}$ is the identity function, the purified problem passed to the solvers of the second loop has exponential size (this is consequently the dominating cost of the whole procedure, see [10] for complexity details). To improve performances, an *incremental* approach is desirable: in the incremental approach, the second loop may be entered before all the possible substitutions have been examined. If the second loop returns unsat, one can exit the whole algorithm and only in case it returns sat further substitutions (producing new conjuncts for ϕ) are taken into consideration. Since when A_I^E-check is called in conclusive steps of our model checking algorithms (for fixpoint, safety, or progress checks), the expected answer is unsat, this incremental strategy may be considerably convenient. Other promising suggestions for reducing the number of instantiations (in the case of particular index and elements theories) come from recent decision procedures for fragments of theories of arrays, see [6,11].

5 Safety Model Checking

Checking safety properties means checking that a set of 'bad states' (e.g., of states violating the mutual exclusion property) cannot be reached by a system. This can be formalized in our settings as follows:

Definition 5.1. *Let $K(a)$ be an $\exists^I$-formula (whose role is that of symbolically representing the set of bad states). The* safety model checking problem *for K is the problem of deciding whether there is an $n \geq 0$ such that the formula*

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge K(a_n) \tag{6}$$

is A_I^E-satisfiable. If such an n does not exist, K is safe*; otherwise, it is* unsafe.

Notice that the A_I^E-satisfiability of (6) means precisely the existence of a finite run leading from a state in I to a state in K.

5.1 Backward Reachability

If a bound for n in the safety model checking is known a priori, then safety can be decided by checking the A_I^E-satisfiability of suitable instances of (6) (this is

possible because each formula (6) is equivalent to a $\exists^{A,I}\forall^{I}$-formula). Unfortunately, this is rarely the case and we must design a more refined method. In the following, we adapt algorithms that incrementally maintain a representation of the set of reachable states in an array-based system.

Definition 5.2. *Let $K(a)$ be an $\exists^{I}$-formula. An $\mathcal{M}$-state s_0 is* backward reachable from K in n steps *iff there exists a sequence of $\mathcal{M}$-states $s_1, \ldots, s_n$ such that*

$$\mathcal{M} \models \tau(s_0, s_1) \wedge \cdots \wedge \tau(s_{n-1}, s_n) \wedge K(s_n).$$

We say that s_0 is backward reachable from K *iff it is backward reachable from K in n steps for some $n \geq 0$.*

Before designing our backward reachability algorithm, we must give a symbolic representation of the set of backward reachable states. Preliminarily, notice that if $K(a)$ is an $\exists^{I}$-formula, the set of states which are backward reachable in one step from K can be represented as follows:

$$Pre(\tau, K) := \exists a' \, (\tau(a, a') \wedge K(a')). \tag{7}$$

Although $Pre(\tau, K)$ is not an $\exists^{I}$-formula anymore, we are capable of finding an equivalent one in the class:

Proposition 5.3. *Let $K(a)$ be an $\exists^{I}$-formula; then $Pre(\tau, K)$ is A_I^E-equivalent to an (effectively computable) $\exists^{I}$-formula $K'(a)$.*

The proof of this Proposition can be obtained by a tedious syntactic computation while using (2) together with the fact that T_E admits quantifier elimination.

Abstractly, the set of states that can be backward reachable from K can be recursively characterized as follows: (a) $Pre^0(\tau, K) := K$ and (b) $Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$. The formula $BR^n(\tau, K) := \bigvee_{s=0}^{n} Pre^s(\tau, K)$ (having just one free variable of type `ARRAY`) symbolically represents the states that can be backward reached from K in n steps. The sequence of $BR^n(\tau, K)$ allows us to rephrase the safety model checking problem (Definition 5.1) using symbolic representations by saying that *K is safe iff the formulae $I \wedge BR^n(\tau, K)$ are all not A_I^E-satisfiable*. This still requires infinitely many tests, *unless there is an n such that the formula $\neg(BR^{n+1}(\tau, K) \rightarrow BR^n(\tau, K))$ is A_I^E-unsatisfiable.*[6] The key observation now is that both $I \wedge BR^n(\tau, K)$ and $\neg(BR^{n+1}(\tau, K) \rightarrow BR^{n+1}(\tau, K))$ can be easily transformed into $\exists^{A,I}\forall^{I}$-formulae so that their A_I^E-satisfiability is decidable and checked by the procedure A_I^E-check of Figure 2.

This discussion suggests the adaptation of a standard backward reachability algorithm (see, e.g., [12]) depicted in Figure 3, where Pre takes a formula, it computes *Pre* as defined in (7), and then applies the required syntactic manipulations to transform the resulting formula into an equivalent one according to Proposition 5.3.

[6] Notice that the A_I^E-unsatisfiability of $\neg(BR^n(\tau, K) \rightarrow BR^{n+1}(\tau, K))$ is obvious by definition. Hence, the A_I^E-unsatisfiability of $\neg(BR^{n+1}(\tau, K) \rightarrow BR^n(\tau, K))$ implies that $A_I^E \models BR^{n+1}(\tau, K) \leftrightarrow BR^n(\tau, K)$.

```
function BReach(K)
    i ⟵ 0; BR^0(τ,K) ⟵ K; K^0 ⟵ K
    if A_I^E-check(BR^0(τ,K) ∧ I) = sat then return unsafe
    repeat
        K^{i+1} ⟵ Pre(τ,K^i)
        BR^{i+1}(τ,K) ⟵ BR^i(τ,K) ∨ K^{i+1}
        if A_I^E-check(BR^{i+1}(τ,K) ∧ I) = sat then return unsafe
        else i ⟵ i+1
    until A_I^E-check(¬(BR^{i+1}(τ,K) → BR^i(τ,K)) = unsat
    return safe
end
```

Fig. 3. Backward Reachability for Array-based Systems

Theorem 5.4. *Let $K(a)$ be an $\exists^I$-formula; then the function* BReach *in Figure 3 semi-decides the safety model checking problem for K.*

Example 5.5. In the Simplified Bakery example of Figure 1 (with the formula K to be tested for safety given like in (1)), the algorithm of Figure 3 stops after four loops and certifies safety of K. ⊣

5.2 Termination of Backward Reachability

Indeed, the main problem with the semi-algorithm of Figure 3 is termination; in fact, it may not terminate when the system is safe. In the literature, termination of infinite state model checking is often obtained by defining a well-quasi-ordering (wqo – see, e.g., [1]) on the configurations of the system. Here, we adapt this approach to our setting, *by defining model-theoretically a notion of configuration and then introducing a suitable ordering on configurations*: once this is done, it will be immediate to prove that termination follows whenever the ordering among configurations is a wqo.

An *A_I^E-configuration* (or, briefly, a configuration) is an $\mathcal{M}$-state in a finite index model $\mathcal{M}$ of A_I^E: a configuration is denoted as $(s, \mathcal{M})$, or simply as s, leaving $\mathcal{M}$ implicit. Moreover, we associate a Σ_I-structure s_I and a Σ_E-structure s_E with an A_I^E-configuration $(s, \mathcal{M})$ as follows: the Σ_I-structure s_I is simply the finite structure $\mathcal{M}_I$, whereas s_E is the Σ_E-substructure of $\mathcal{M}_E$ generated by the image of s (in other words, if $\texttt{INDEX}^{\mathcal{M}} = \{c_1, \ldots, c_k\}$, then s_E is generated by $\{s(c_1), \ldots, s(c_k)\}$).

We remind few definitions about preorders. A preorder $(P, \leq)$ is a set endowed with a reflexive and transitive relation; an *upset* of such a preorder is a subset $U \subseteq P$ such that ($p \in U$ and $p \leq q$ imply $q \in U$). An upset U is *finitely generated* iff it is a finite union of cones, where a *cone* is an upset of the form $\uparrow p = \{q \in P \mid p \leq q\}$ for some $p \in P$. A preorder $(P, \leq)$ is a *well-quasi-ordering* (wqo) iff every upset of P is finitely generated (this is equivalent to the standard definition, see [10]). We define now a preorder among our configurations:

Definition 5.6. *Let s, s' be configurations; $s' \leq s$ holds iff there are a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that the set-theoretical compositions of μ with s and of s' with ν are equal.*

Let $K(a)$ be an $\exists^I$-formula: we denote by $[\![K]\!]$ the set of A^E_I-configurations satisfying K; in symbols, $[\![K]\!] := \{(\mathcal{M}, s) \mid \mathcal{M} \models K(s)\}$.

Proposition 5.7. *For every $\exists^I$-formula $K(a)$, the set $[\![K]\!]$ is upward closed; also, for every $\exists^I$-formulae K_1, K_2, we have $[\![K_1]\!] \subseteq [\![K_2]\!]$ iff $A^E_I \models K_1 \rightarrow K_2$.*

The set of configurations $\mathcal{B}(\tau, K)$ which are backward reachable from a given $\exists^I$-formula K is thus an upset, being the union of infinitely many upsets; however, even in case the latter are finitely generated, $\mathcal{B}(\tau, K)$ needs not be so. Under the hypothesis of local finiteness of T_E, this is precisely what characterizes termination of backward reachability search:

Theorem 5.8. *Assume that T_E is locally finite; let K be an $\exists^I$-formula. If K is safe, then* BReach *in Figure 3 terminates iff $\mathcal{B}(\tau, K)$ is a finitely generated upset.*[7] *As a consequence,* BReach *always terminates when the preorder on A^E_I-configurations is a wqo.*

The termination result of Theorem 5.8 covers many well-known special cases, like broadcast protocols, some versions of the bakery algorithm, lossy channel systems (notice that for the latter, there doesn't exist a primitive recursive complexity lower bound); to apply Theorem 5.8, one needs classical facts like Dikson's Lemma, Higman's Lemma, Kruskal's Theorem, etc. (see again [10] for details).

6 Progress Formulae for Recurrence Properties

Liveness problems are difficult and even more so for infinite state systems. In the following, we consider a special kind of liveness properties, which falls into the class of recurrence properties in the classification introduced in [13]. Our method for dealing with such properties consists in synthesizing progress functions definable at quantifier free level.

A recurrence property is a property which is *infinitely often* true in every infinite run of a system; in other words, to check that R is a recurrence property it is sufficient to show that it cannot happen that there is an infinite run $s_0, s_1, \ldots, s_i, \ldots$ such that R is true at most in the states from a finite prefix $s_0, \ldots, s_m$. This can be formalized in our framework as follows.

Definition 6.1. *Suppose that $R(a)$ is a $\forall^I$-formula; let us use the abbreviations $K(a)$ for $\neg R(a)$ and $\tau_K(a, a')$ for $\tau(a, a') \wedge K(a) \wedge K(a')$.*[8] *We say that R is a* recurrence property *iff for every m the infinite set of formulae*

$$I(b_1), \tau(b_1, b_2), \ldots, \tau(b_m, a_0), \tau_K(a_0, a_1), \tau_K(a_1, a_2), \ldots \tag{8}$$

[7] If K is unsafe, we already know that BReach terminates because it detects unsafety (cf. Theorem 5.4).

[8] Notice that τ_K is easily seen to be equivalent to a disjunction of T-formulae, like the original τ.

is not satisfiable in a finite index model of A_I^E.

The finite prefix $I(b_1), \tau(b_1, b_2), \ldots, \tau(b_m, a_0)$ in (8) above ensures that the states assigned to $a_0, a_1, \ldots$ are (forward) reachable from an initial state, whereas the infinite suffix $\tau_K(a_0, a_1)$, $\tau_K(a_1, a_2), \ldots$ expresses the fact that property R is never attained. Observe that, whereas it can be shown that the A_I^E-satisfiability of (6) for safety problems (cf. Definition 5.1) is equivalent to its satisfiability in a finite index model of A_I^E, this is not the case for (8). This imposes the above explicit restriction to finite index models since, otherwise, recurrence might unnaturally fail in many concrete situations.

Example 6.2. In the Simplified Bakery example of Figure 1, we take K to be

$$\exists i \; (i = d \wedge a[i] = \texttt{wait})$$

(here d is a fresh constant of type Σ_I). Then the A_I^E-satisfiability of (8) implies that the protocol cannot guarantee absence of starvation. ⊣

Let R be a recurrence property; we say that R has *polynomial complexity* iff there exists a polynomial $f(n)$ such that for every m and for $k > f(n)$ the formulae

$$I(b_1) \wedge \tau(b_1, b_2) \wedge \cdots \wedge \tau(b_m, a_0) \wedge \tau_K(a_0, a_1) \wedge \cdots \wedge \tau_K(a_{k-1}, a_k) \qquad (9)$$

are all unsatisfiable in the models $\mathcal{M}$ of A_I^E such that the cardinality of the support of $\mathcal{M}_I$ is at most n (in other words, $f(n)$ gives an upper bound for the waiting time needed to reach R by a system consisting of less than n processes).

Definition 6.3. *Let $R(a)$ be a $\forall^I$-formula and let K and τ_K be as in Definition 6.1. A formula $\psi(\underline{t}(\underline{j}), a[\underline{t}(\underline{j})])$ is an* index invariant *iff there exists a safe $\exists^I$-formula H such that*

$$A_I^E \models \forall a_0 \; \forall a_1 \; \forall \underline{j} \; (\neg H(a_0) \wedge \tau_K(a_0, a_1) \rightarrow (\psi(\underline{t}, a_0[\underline{t}]) \rightarrow \psi(\underline{t}, a_1[\underline{t}]))). \qquad (10)$$

A progress condition for *R is a finite set of index invariant formulae*

$$\psi_1(\underline{t}(\underline{j}), a[\underline{t}(\underline{j})]), \ldots, \psi_c(\underline{t}(\underline{j}), a[\underline{t}(\underline{j})])$$

for which there exists a safe $\exists^I$-formula H such that

$$A_I^E \models \forall a_0 \, \forall a_1 \, \big(\neg H(a_0) \wedge \tau_K(a_0, a_1) \rightarrow \exists \underline{j} \bigvee_{k=1}^{c} (\neg \psi_k(\underline{t}, a_0[\underline{t}]) \wedge \psi_k(\underline{t}, a_1[\underline{t}]))\big). \quad (11)$$

Suppose there is a progress condition as above for R; if $\mathcal{M}$ is a model of A_I^E such that the support of $\mathcal{M}_I$ has cardinality at most n, then the formula (9) cannot be satisfiable in $\mathcal{M}$ for $k > c \cdot n^\ell$ (here ℓ is the length of the tuple $\underline{j}$). This argument shows the following:

Proposition 6.4. *If there is a progress condition for R, then R is a recurrence property with polynomial complexity.*

Example 6.5. In the Simplified Bakery example of Example 6.2, the following four index invariant formulae:

$$a[j].s = \texttt{crash} \qquad a[j].t = 0 \ \vee \ a[j].t > a[d].t$$
$$a[j].t > a[d].t \qquad \neg(a[j].t < a[d].t \ \wedge \ a[j].s = \texttt{wait})$$

constitute a progress condition, thus guaranteeing that each non faulty process d can get (in polynomial time) the resource every time it asks for it. The algorithm of Figure 3 is needed for certifying safety of some formulae (like $\exists i\,(a[i].s = \texttt{use} \wedge a[i].t = 0)$) to be used as the H of Definition 6.3. ⊣

Let us now discuss how to mechanize the application of Proposition 6.4. The validity tests (10) and (11) can be reduced to unsatisfiability tests covered by Theorem 4.1; however, the search space for progress conditions looks to be unbounded for various reasons (the number of the ψ's of Definition 6.3, the length of the string $\underline{j}$, the 'oracle' safe H's, etc.). Nevertheless, the proof of the following unexpected result shows (in the appropriate hypotheses) how to find progress conditions whenever they exist:

Theorem 6.6. *Suppose that T_E is locally finite and that safety of $\exists^I$-formulae can be effectively checked. Then the existence of a progress condition for a given $\forall^I$-formula R is decidable.*

7 Related work and Conclusions

A popular approach to uniform verification (e.g., [2,3,4]) consists of defining a finitary representation of the (infinite) set of states and then to explore the state space by using such a suitable data structure. Although such a data structure may contain declarative information (e.g., constraints over some algebraic structure or first-order formulae), a substantial part is non-declarative. Typically, the part of the state handled non-declaratively is that of indexes (which, for parametrized systems, specify the topology), thereby forcing to re-design the verification procedures whenever their specification is changed (for the case of parametrized systems, whenever the topology of the systems changes). Since our framework is fully declarative, we avoid this problem altogether.

There has been some attempts to use a purely logical techniques to check both safety and liveness properties of infinite state systems (e.g., [7,9]). The hard part of these approaches is to guess auxiliary assertions (either invariants, for safety, or ranking functions, for liveness). Indeed, finding such auxiliary assertions is not easy and automation requires techniques adapted to specific domains (e.g., integers). In this paper, we have shown how our approach does not require auxiliary invariants for safety and proved that the search for certain progress conditions can be fully mechanized (Theorem 6.6).

In order to test the viability of our approach, we have built a prototype implementation of the backward reachability algorithm of Figure 3. The function A_I^E-check has been implemented by using a naive instantiation algorithm and

a state-of-the-art SMT solver. We have also implemented the generation of the proof obligations (10) and (11) for recognizing progress conditions. This allowed us to automatically verify the examples discussed in this paper and in [10]. We are currently planning to build a robust implementation of our techniques.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite state systems. In *Proc. of LICS '96*, pages 313–321, 1996.
2. P. A. Abdulla, G. Delzanno, N. Ben Henda, and A. Rezine. Regular model checking without transducers. In *Proc of TACAS '07*, volume 4424 of *LNCS*, pages 721–736. Springer, 2007.
3. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *Proc. of CAV '07*, volume 4590 of *LNCS*, pages 145–157. Springer, 2007.
4. A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Proc. of FCT '07*, volume 4639 of *LNCS*, pages 1–22. Springer, 2007.
5. M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient theory combination via boolean search. *Information and Computation*, 204(10):1493–1525, 2006.
6. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *Proc. of VMCAI '06*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.
7. L. M. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Proc. of CADE '02*, volume 2392 of *LNCS*, pages 438–455. Springer, 2002.
8. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York-London, 1972.
9. Y. Fang, N. Piterman, A. Pnueli, and L. D. Zuck. Liveness with invisible ranking. *Software Tools for Technology*, 8(3):261–279, 2006.
10. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT model checking of array-based systems. Technical Report RI318-08, Università degli Studi di Milano, 2008. Available at `http://homes.dsi.unimi.it/~zucchell/publications/techreport/GhiNiRaZu-RI318-08.pdf`.
11. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *Proc. of TACAS '08*, volume 4963 of *LNCS*, pages 265–281. Springer, 2008.
12. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256(1-2):93–112, 2001.
13. Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. of PODC '90*, pages 377–410. ACM Press, 1990.
14. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
15. T. Rybina and A. Voronkov. A logical reconstruction of reachability. In *Proc. of PSI '03*, volume 2890 of *LNCS*, pages 222–237. Springer, 2003.
16. R. Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.