Light-Weight SMT-based Model Checking

Silvio Ghilardi[1]

Dip. di Scienze dell'Informazione, Univ. di Milano (Italia)

Silvio Ranise[2]

Dip. di Informatica, Univ. di Verona (Italia)

Thomas Valsecchi[3]

Dip. di Scienze dell'Informazione, Univ. di Milano (Italia)

Abstract

Recently, the notion of an array-based system has been introduced as an abstraction of infinite state systems (such as mutual exclusion protocols or sorting programs) which allows for model checking of invariant (safety) and recurrence (liveness) properties by Satisfiability Modulo Theories (SMT) techniques. Unfortunately, the use of quantified first-order formulae to describe sets of states makes fix-point checking extremely expensive. In this paper, we show how invariant properties for a sub-class of array-based systems can be model-checked by a backward reachability algorithm where the length of quantifier prefixes is efficiently controlled by suitable heuristics. We also present various refinements of the reachability algorithm that allows it to be easily implemented in a client-server architecture, where a "light-weight" algorithm is the client generating proof obligations for safety and fix-point checks and an SMT solver plays the role of the server discharging the proof obligations. We also report on some encouraging preliminary experiments with a prototype implementation of our approach.

Keywords: Model Checking of Infinite State Systems, Satisfiability Modulo Theories, Safety

1 Introduction

An integration of Satisfiability Modulo Theories (SMT) solving in a backward reachability algorithm has been proposed in [22] for the model checking of invariant (safety) properties of a large class of infinite state systems—called, *array-based systems*. Roughly, an array-based system is a transition system which updates one (or more) array variable a. Being parametric in the structures associated to the indexes and the elements in a, the notion of array-based system is quite flexible and allows one to specify a large of class of infinite state systems. For example, consider

[1] Email: ghilardi@dsi.unimi.it

[2] Email: silvio.ranise@univr.it

[3] Email: thomas.valsecchi@gmail.com

parametrised systems and the task of specifying their topology: by using no structure at all, indexes are simply identifiers of processes that can only be compared for equality; by using a linear order, indexes are identifiers of processes so that it is possible to distinguish between those on the left or on the right of a process with a particular identifier; by using richer and richer structures (such as trees and graphs), it is possible to specify more and more complex topologies. Similar observations hold also for elements, where it is well-known how to use algebraic structures to specify data structures. Formally, the structure on both indexes and elements is declaratively and uniformly specified by *theories*, i.e. pairs formed by a (first-order) language and a class of (first-order) structures.

In this framework, invariant properties of array-based systems can be verified by using a symbolic version of a *backward search algorithm* which repeatedly computes the *pre-image* of the set of states from which it is possible to reach the set of *unsafe* states, i.e. the states violating the desired invariant property. The algorithm halts in two cases, either when the current set of reachable states has a non-empty intersection with the set of initial states and the system is unsafe, or when the current set of reachable states has reached a fix-point (i.e. further application of the transition does not enlarge the set of reachable states) and the system is safe. To mechanize this approach, the following three requirements are mandatory: (i) the class $\mathcal{F}$ of (possibly quantified) first-order formulae is expressive enough to represent sets of states and invariants, (ii) $\mathcal{F}$ is closed under pre-image computation, and (iii) the checks for safety and fix-point can be reduced to decidable logical problems (e.g., satisfiability) of formulae in $\mathcal{F}$. Once requirements (i)—(iii) are satisfied, this technique can be seen as a symbolic version of the model checking techniques of [8] revisited in the declarative framework of first-order logic augmented with theories [22]. Using this declarative framework has several *potential* advantages; two of the most important ones are the following. First, the computation of the pre-image (cf. requirement (ii) above) becomes computationally cheap: we only need to build the formula ϕ representing the (iterated) pre-images of the set of unsafe states and then put the burden of using suitable data structures to represent ϕ on the available (efficient) solver for logical problems encoding safety and fix-point checks. This is in sharp contrast to what is usually done in almost all other approaches to symbolic model checking of infinite state systems, where the computation of the pre-image is computationally very expensive as it requires a substantial process of normalization on the data structure representing the (infinite) sets of states so as to simplify safety and fix-point checks.

The second advantage is the possibility to use state-of-the-art SMT solvers, a technology that is showing very good success in scaling up various verification techniques, to support both safety and fix-point checks (cf. requirement (iii) above). Unfortunately, the kind of satisfiability problems obtained in the context of the backward search algorithm requires to cope with (universal) quantifiers and this makes the off-the-shelf use of SMT solvers problematic. In fact, even when using classes of formulae with decidable satisfiability problem, currently available SMT solvers are not yet mature enough to efficiently discharge formulae containing (universal) quantifiers, despite the fact that this problem has recently attracted a lot of efforts (see, e.g., [17,21,15]). To alleviate this problem, we have designed a general decision procedure for a class of formulae satisfying requirement (i) above, based on quantifier instantiation (see [22] and Theorem 3.4 below); this allows for an easier way to integrate currently available SMT-solvers in the backward search algorithm. Unfortunately, the number of instances required by the instantiation algorithm is still very large and preliminary experiments have shown unacceptable performances. This fact together with the observation that the size of the formulae generated by the backward search algorithm grows very quickly demand a principled approach to the pragmatics of efficiently integrating SMT solvers in the backward search algorithm. In this respect, the paper makes two important contributions.

We focus on a sub-class of the (quantified) formulae in [22] (Section 3) to model a smaller but still significant class of systems analogous to the well-known guarded assignment systems (see, e.g., [28]). Our *first contribution* (Section 4.1) is to find sufficient conditions under which, it is correct to *reduce* (and sometimes also to eliminate) the quantifiers in the formulae representing (iterated) pre-images. The *second contribution* (Section 4.2) is a discussion about the of how to adapt implementation techniques, known in the field of symbolic model checking, to the backward search algorithm so that a client-server architecture can be used, where a "light-weight" client (i.e. a program with few lines of code) generates proofs obligation for fix-point and safety checks for an SMT solver, the server. Preliminary experiments seem to confirm the viability and scalability of the approach. (For lack of space, technical details are in the Appendix).

2 Preliminaries

We assume the usual syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, sub-structure, truth, satisfiability, and validity) notions of first-order logic (see, e.g., [20]). The equality symbol $=$ is included in all signatures considered below. A signature is *relational* if it does not contain function symbols and it is *quasi-relational* if its function symbols are all (individual) constants. An *expression* is a term, an atom, a literal, or a formula. Let $\underline{x}$ be a finite tuple of variables and Σ a signature, a $\Sigma(\underline{x})$-expression is an expression built out of the symbols in Σ where at most the variables in $\underline{x}$ may occur free (we will write $E(\underline{x})$ to emphasize that E is a $\Sigma(\underline{x})$-expression).

Satisfiability Modulo Theory. According to the current practice in the SMT literature [24], a *theory* T is 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 the *models* of T. Below, we let $T = (\Sigma, \mathcal{C})$.[4] A Σ-formula ϕ is *T-satisfiable* if there exists a Σ-structure $\mathcal{M}$ in $\mathcal{C}$ such that ϕ is true in $\mathcal{M}$ under a suitable assignment to the free variables of ϕ (in symbols, $\mathcal{M} \models \phi$); it is *T-valid* (in symbols, $T \models \varphi$) if its negation is T-unsatisfiable. Two formulae φ_1 and φ_2 are *T-equivalent* if $\varphi_1 \leftrightarrow \varphi_2$ is *T-valid*. The *satisfiability modulo the theory T ($SMT(T)$) problem* amounts to establishing the T-satisfiability of quantifier-free (i.e. not containing quantifiers) Σ-formulae. A *theory solver* for T (T-solver) is any

[4] An important class of theories, ubiquitously used in verification, formalizes enumerated data types. An *enumerated data-type theory* T is a theory in a quasi-relational signature whose class of models contains only a single finite Σ-structure $\mathcal{M} = (M, \mathcal{I})$ such that for every $m \in M$ there exists a constant $c \in \Sigma$ such that $c^{\mathcal{I}} = m$. Below, we use enumerated data-type theories to model control locations of processes in parameterized systems.

procedure capable of establishing whether any given finite conjunction of Σ-literals is T-satisfiable or not. The *lazy approach* to solve $SMT(T)$ problems consists of integrating a DPLL Boolean enumerator with a T-solver (see, e.g., [29] for details).

Definitional extension of a theory. Below, for technical reasons, it will be useful to extend theories with functions in a constrained way. A (quantifier-free) *T-definable function* is a quantifier-free formula $\phi(\underline{x}, y)$ such that

$$T \models \forall \underline{x} \exists y\, \phi(\underline{x}, y) \quad \text{and} \quad T \models \forall \underline{x} \forall y_1 \forall y_2\, (\phi(\underline{x}, y_1) \land \phi(\underline{x}, y_2) \to y_1 = y_2).$$

A *definable extension* $T' = (\Sigma', \mathcal{C}')$ of a theory $T = (\Sigma, \mathcal{C})$ is obtained from T by applying—finitely many times—the following procedure: (i) consider a T-definable function $\phi(\underline{x}, y)$; (ii) let $\Sigma' := \Sigma \cup \{F\}$, where F is a "fresh" function symbol (i.e. $F \notin \Sigma$) whose arity is equal to the length of $\underline{x}$; (iii) take as $\mathcal{C}'$ the class of Σ'-structures $\mathcal{M}$ whose Σ-reduct is a model of T and such that

$$\mathcal{M} \models \forall \underline{x} \forall y\, (F(\underline{x}) = y \leftrightarrow \phi(\underline{x}, y)).$$

Indeed, the $SMT(T')$ problem for such a T' can be solved by replacing the new function symbols with fresh constants, adding their definitions as conjuncts to the formula to be tested for satisfiability, and invoking a solver for $SMT(T)$.

In the following, we adopt a many-sorted version of first-order logic. All notions introduced above can be easily adapted to this setting (see again [20]).

3 Model-Checking of Array-based Systems

We consider the formalism of *guarded assignment* array-based systems, a restricted version of that defined in [22]. We focus on *parametrised systems*, i.e. systems consisting of an arbitrary (but finite) number of identical processes, since a large number of such systems can be expressed in this formalism. There exist two kinds of guards, expressing *existential* or *universal global conditions*, on the state of a parameterized system. As we will see, while existential conditions can be directly expressed in our formalism, universal conditions require us to model parameterized systems following the so-called *stopping failures model* for distributed algorithms [23], which is quite close to the *approximate* model of [9,10]. The key property of a parameterized system modelled according to the stopping failures model is that processes may fail without warning at any time. To formalize this, assume that a process in a parametrised system has a finite set $Q = \{q_1, ..., q_n\}$ of control locations plus other local data variables. Then, consider an extended set $Q' = Q \cup \{q_{crash}\}$, where $q_{crash} \notin Q$, and augment the set of transitions of each process as follows: it is always possible to go from state q_i to q_{crash}, for each $i = 1, ..., n$. An example of universal global condition is a guard saying that a process i can execute a transition if a certain condition C is satisfied by *all* processes $j \neq i$. In the stopping failures model, this can be expressed without the universal quantification as follows: the process i takes the transition without checking the global condition C and, concurrently, all processes $j \neq i$ not satisfying the condition C move to the state q_{crash}; moreover, all processes $j \neq i$ satisfying C behave as originally prescribed. The stopping failures model of the system satisfies a sub-set of the class of safety (or even recurrence) properties satisfied by the original system (since the latter has fewer runs), hence establishing a safety property for the stopping failures model *implies* that the same property is enjoyed by the original system.

Example 3.1 Consider the simplified variant of the Bakery algorithm of [10], where a finite (but unknown) number of processes should be granted mutual exclusion to a critical section by using tickets. Processes are arranged in an array whose indexes are linearly ordered and each process can be in one of three states: idle, wait, crit(ical). At the beginning, all processes are in the idle state. There are three possible transitions involving a single process with index z (in all transitions, the processes with index different from z remain in the same state): (τ_1^o) z is in idle, all the processes to its left are idle, and z moves to wait; (τ_2^o) z is in wait, all the processes to its right are idle, and z moves to critical; and (τ_3^o) z is in crit and moves to idle. The system should satisfy the following mutual exclusion property: there are no two distinct processes in crit at the same time.

Since we adopt the stopping failures model, we introduce an additional state crash and three additional transitions: (τ_x^c) if a process with index z is in state x, then it moves to crash and all the other processes remain in the same state (for $x \in \{$ idle, wait, crit $\}$). The transitions τ_1^o, τ_2^o, and τ_3^o are transformed as follows: (τ_1) if a process with index z is idle, then it moves to wait; furthermore, any process on its left remains in the same state and for any process on its left if the process is not idle, then it moves to crash, otherwise it remains idle; (τ_2) if a process with index z is in wait, then it moves to critical; furthermore, any process on its left remains in the same state and for any process on its right it is not idle, then it moves to crash, otherwise it remains idle; and (τ_3) if a process with index z is in crit, then it becomes idle and all other processes remains in the same state. The new system is supposed to satisfy the same mutual exclusion property of the original system above.

In the following, we use the term "running example" to indicate the stopping failures model of this system. When discussing the application of our verification techniques to the running example, we forget the transitions (τ_x^c), for $x \in \{$ idle, wait, crit $\}$, since their structure is similar to (τ_3) and all observations for the latter apply trivially to the former. □

Theories for indexes and elements. The state of an array-based system consists of a single array (however, it is straightforward to generalize all definitions below to the case of several arrays) indexed by a data structure I (e.g., by a finite and linearly ordered set of identifiers), storing elements of a data structure E (e.g., an enumerated data type for the control locations). To formalize this in our declarative formalism, we use two theories: T_I for indexes (intuitively, the role of T_I is to specify the "topology" of the system) and T_E for data (the role of T_E is to specify the set of values over which local data variables values range). **In the rest of the paper, we fix** (i) a theory $T_I = (\Sigma_I, \mathcal{C}_I)$ whose only sort symbol is `INDEX`; (ii) a theory $T_E = (\Sigma_E, \mathcal{C}_E)$ whose only sort symbol is `ELEM` (the class of models $\mathcal{C}_E$ of this theory is usually reduced to a single structure).

The **theory** $A_I^E = (\Sigma, \mathcal{C})$ **of arrays with indexes** I **and elements** E is obtained as the combination of T_I and T_E as follows: `INDEX`, `ELEM`, and `ARRAY` are the only sort symbols of A_I^E, the signature is $\Sigma := \Sigma_I \cup \Sigma_E \cup \{\_[\_]\}$ where $\_[\_] : \mathtt{ARRAY}, \mathtt{INDEX} \longrightarrow \mathtt{ELEM}$ (intuitively, $a[i]$ denotes the element stored in the array a at index i); a three-sorted structure $\mathcal{M} = (\mathtt{INDEX}^{\mathcal{M}}, \mathtt{ELEM}^{\mathcal{M}}, \mathtt{ARRAY}^{\mathcal{M}}, \mathcal{I})$ is in $\mathcal{C}$ iff $\mathtt{ARRAY}^{\mathcal{M}}$ is the set of (total) functions from $\mathtt{INDEX}^{\mathcal{M}}$ to $\mathtt{ELEM}^{\mathcal{M}}$, the function symbol $\_[\_]$ is interpreted as function application, and $\mathcal{M}_I = (\mathtt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$, $\mathcal{M}_E = (\mathtt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ are models of T_I and T_E, respectively (where $\mathcal{I}_{|\Sigma_X}$ is the restriction of the interpretation $\mathcal{I}$ to the symbols in Σ_X for $X \in \{I, E\}$).

Example 3.2 To begin the formalization of the running example, we take T_I to be the theory of finite and linearly ordered sets: the signature Σ_I is relational and contains only the binary predicate $<$. Furthermore, let T_E be the enumerated data type theory whose signature contains a constant for each of the four possible control locations: `idle`, `wait`, `crit`, and `crash` (hence Σ_E is quasi-relational). □

Array-based systems. A *(guarded assignment) array-based (transition) system (for* (T_I, T_E)*)* is a triple $\mathcal{S} = (a, I, \tau)$ where (i) a is the *state* variable of sort `ARRAY`; (ii) $I(a)$ is the *initial* $(\Sigma \cup \Sigma_D)(a)$-formula; and (iii) $\tau(a, a')$ is the *transition* $(\Sigma \cup \Sigma_D)(a, a')$-formula, where a' is a renamed copy of a and Σ_D is the set of defined function symbols not in $\Sigma_I \cup \Sigma_E$. Below, for the sake of simplicity, any definable extension of A_I^E will still be denoted with A_I^E. In making such a definitional extension, we always assume to use defining formulae $\phi(\underline{x}, y)$ such that ϕ *is* *a quantifier-free* $(\Sigma_I \cup \Sigma_E)$*-formula and the variable* y *is of sort* ELEM.

Example 3.3 Let T_I and T_E be as in Example 3.2 and $\Sigma_D := \{F^1, F^2, F^3\}$. The array-based transition system for the Bakery algorithm can be specified as follows (for simplicity, we omit sorts):

$$I(a) := \forall i.a[i] = \mathtt{idle} \qquad \text{and} \qquad \tau(a,a') := \bigvee_{i=1}^{3} \exists z.\phi_L^i(z,a[z]) \wedge \forall j.a'[j] = F^i(z,a[z],j,a[j]),$$

where $\phi_L^i(z,a[z]) := (a[z] = x_i)$ for $i = 1,2,3$, x_1 is idle, x_2 is wait, x_3 is crit, and

$$F^1(z,a[z],j,a[j]) := \begin{cases} \mathtt{wait} & \text{if } j = z \\ a[j] & \text{if } j < z \\ a[j] & \text{if } j > z \wedge a[j] = \mathtt{idle} \\ \mathtt{crash} & \text{otherwise} \end{cases} \qquad F^3(z,a[z],j,a[j]) := \begin{cases} \mathtt{idle} & \text{if } j = z \\ a[j] & \text{otherwise} \end{cases}$$

$$F^2(z,a[z],j,a[j]) := \begin{cases} \mathtt{crit} & \text{if } j = z \\ a[j] & \text{if } j > z \\ a[j] & \text{if } j < z \wedge a[j] = \mathtt{idle} \\ \mathtt{crash} & \text{otherwise} \end{cases}$$

For the sake of clarity, the functions F^i's are defined by cases; it is a trivial exercise to formalize them in extensions of first-order logic supporting an '**if then else**' term constructor as it is customary in SMT solving [24]. Notice that the negation of the mutual exclusion property can be formalized as $K(a) := \exists z_1, z_2.(z_1 \neq z_2 \wedge a[z_1] = \mathtt{crit} \wedge a[z_2] = \mathtt{crit})$. □

Backward Reachability. Given an array-based transition system $\mathcal{S} = (a, I, \tau)$, many symbolic model-checking algorithms are based on computing the set $BR^n(\tau, K)$ of *backward reachable states*, starting from a formula $K(a)$ describing a set of unsafe states. The set $BR^n(\tau, K)$ can be found by iteratively computing the set of backward reachable state in one step, i.e.

(1) $Pre(\tau, K) := \exists a'.(\tau(a,a') \wedge K(a'))$.

Then, $BR^n(\tau, K) := \bigvee_{s=0}^{n} Pre^s(\tau, K)$, where

$$Pre^0(\tau, K) := K \quad \text{and} \quad Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K)).$$

This iteration reaches a fix-point at $n+1$ iff $BR^{n+1}(\tau, K) \to BR^n(\tau, K)$ is A_I^E-valid. Furthermore, if $BR^{n+1}(\tau, K) \wedge I$ is A_I^E-unsatisfiable, then $\mathcal{S}$ is *safe* (w.r.t. K); otherwise, it is *unsafe*.

In order to be able to exploit the backward reachability algorithm sketched above to check invariant properties, it is mandatory to identify a class of first-order formulae such that it should be possible to: (R1) express I, τ, and K for a large number of (abstractions of) systems; (R2) check both A_I^E-satisfiability and A_I^E-validity for the safety and fix-point tests described above, respectively; and (R3) compute a formula which is logically equivalent to $Pre(\tau, K)$ and which is of the same shape as K (this will make the fulfillment of (R2) easier).

Formulae for states and transitions. Intuitively, the class of formulae satisfying (R1) contains those used in Example 3.3. To make this observation precise, we introduce some notational conventions: d, e range over variables of sort ELEM, a over variables of sort ARRAY, $i, j, k, z, \ldots$ over variables of sort INDEX. An underlined variable name abbreviates a tuple of variables of unspecified (but finite) length and, if $\underline{i} := i_1, \ldots, i_n$, the notation $a[\underline{i}]$ abbreviates the tuple of terms $a[i_1], \ldots, a[i_n]$. Possibly sub/super-scripted expressions of the form $\phi(\underline{i}, \underline{e}), \psi(\underline{i}, \underline{e})$ denote *quantifier-free* $(\Sigma_I \cup \Sigma_E \cup \Sigma_D)$*-formulae in which at most the variables* $\underline{i} \cup \underline{e}$ *occur* (notice in particular that no array variable and no constructor $\_[\_]$ can occur here). Also, $\phi(\underline{i}, \underline{t}/\underline{e})$ (or simply $\phi(\underline{i}, \underline{t})$) abbreviates the substitution of the terms $\underline{t}$ for the variables $\underline{e}$

(here, the constructor $\_[\_]$ may appear in t). Thus, for instance, $\phi(\underline{i}, a[\underline{i}])$ denotes the formula obtained by replacing $\underline{e}$ with $a[\underline{i}]$ in the quantifier-free formula $\phi(\underline{i}, \underline{e})$. An $\exists^I$*-formula* is a formula of the form $\exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$ (see, e.g., the formula $K(a)$ in Example 3.3). A $\forall^I$*-formula* is a formula of the form $\forall \underline{i}.\phi(\underline{i}, a[\underline{i}])$ (see, e.g., the formula $I(a)$ in Example 3.3).

According to [22], a transition can be split into a local and a global component. In the restricted format adopted in this paper, the *local component* is a guard expressing a condition to be satisfied by a finite number of indexes and the *global component* is a deterministic update of the whole system which is represented by a definable function. Formally, let $\phi_L(\underline{i}, \underline{e})$ be a quantifier-free formula and $F(\underline{i}, \underline{e}, j, d)$ be a defined function symbol. A *T-formula* with guard ϕ_L and global update F is a formula of the form

$$\exists \underline{i}\ (\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall j\ a'[j] = F(\underline{i}, a[\underline{i}], j, a[j])). \tag{2}$$

In the rest 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**. An example of such a system is in Example 3.3: the ϕ^i_L's are local components, the F^i's are global updates, and the transition τ is a disjunction of three T-formulae.

Satisfiability checking. Concerning (R2), recall the formulae that we are supposed to use for the safety and fix-points checks: $I \wedge BR^n(\tau, K)$ and $\neg(BR^{n+1}(\tau, K) \to BR^n(\tau, K))$, where the latter is negated since we reason by refutation as we use only SMT solvers. Under the hypothesis (verified below) of closure under pre-image computation—cf. (R3)—both formulae above are of the form $\exists a\ \exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, a[\underline{i}], a[\underline{j}])$. Following [22], such formulae are called $\exists^{A,I}\forall^I$*-sentences.*

Theorem 3.4 ([22]) *The A^E_I-satisfiability of $\exists^{A,I}\forall^I$-sentences is decidable if (i) T_I has a quasi-relational signature and it is closed under substructures; (ii) the $SMT(T_I)$ and $SMT(T_E)$ problems are decidable.* [5]

The peculiarity of the above result (when compared with similar ones available in literature, e.g., [12]) is the *model-theoretic nature* of the conditions on the *parametric* input theory T_I that ensure decidability. The (proof of this) Theorem (see [22]) suggests the following quantifier instantiation algorithm: first, eliminate the universal quantifiers of $\exists^{A,I}\forall^I$-sentences by instantiating the $\underline{j}$'s with the constants in Σ_I and the $\underline{i}$'s, considered as (Skolem) constants, in all possible ways; then, invoke the available SMT solver for A^E_I. The decidability of the $SMT(A^E_I)$ problem can be shown by using *generic combination techniques* from the decidability of those for $SMT(T_I)$ and $SMT(T_E)$ (see [22] for details). From now on, **we assume that the theories T_I and T_E always satisfy the hypotheses of Theorem 3.4**.

Closure under pre-image. Condition (R3) is ensured by the following result.

Proposition 3.5 *Let $K(a)$ be an $\exists^I$-formula; then $Pre(\tau, K)$ is A^E_I-equivalent to an (effectively computable) $\exists^I$-formula.*

[5] Notice that hypothesis (i) is satisfied in all practical cases (when, e.g., the models of T_I are all (finite) sets, linear orders, graphs, forests, etc.); the hypothesis that T_I has a quasi-relational signature can be weakened to local finiteness as in [22]. Quantifier elimination for T_E is assumed in [22] to show closure under pre-image computation: here we do not need it, as we adopt a more restricted notion for T-formulae.

function $\mathsf{BReach}(K)$
  $i \longleftarrow 0$; $BR^0(\tau, K) \longleftarrow K$; $K^0 \longleftarrow K$
  if $BR^0(\tau, K) \wedge I$ is A_I^E-sat. **then return** unsafe
  repeat
    $K^{i+1} \longleftarrow \mathsf{Pre}(\tau, K^i)$
    $BR^{i+1}(\tau, K) \longleftarrow BR^i(\tau, K) \vee K^{i+1}$
    if $BR^{i+1}(\tau, K) \wedge I)$ is A_I^E-sat. **then return** unsafe
    else $i \longleftarrow i + 1$
  until $\neg(BR^{i+1}(\tau, K) \to BR^i(\tau, K))$ is A_I^E-unsat.
  return safe
end

Fig. 1. Backward Reachability Algorithm

Proof. Let $K(a) := \exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ and $\tau(a, a') := \bigvee_{h=1}^{s} \exists \underline{i}\, (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \forall j\; a'[j] = F^h(\underline{i}, a[\underline{i}], j, a[j]))$. Now, $\forall j\; a'[j] = F^h(i, a[i], j, a[j])$ can be equivalently rewritten as $a' = \lambda j. F^h(\underline{i}, a[\underline{i}], j, a[j])$ using λ-abstraction. Thus, if we eliminate the quantifier $\exists a'$ and then apply β-conversion, we get that $Pre(\tau, K)$ is equivalent to

$$(3) \quad \exists \underline{i}\, \exists \underline{k} \bigvee_{h=1}^{s} \; (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$$

where $\phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))]$ is the formula obtained from $\phi(\underline{k}, a'[\underline{k}])$ by replacing $a'[k_m]$ with $F^h(\underline{i}, a[\underline{i}], k_m, a[k_m])$ for $m = 1, ..., l$ (here $\underline{k}$ is the tuple $k_1, \ldots, k_l$). $\square$

As suggested by the proof of Proposition 3.5, the implementation of Pre simply amounts to build up formula (3): the task of simplifying it by eliminating redundancies is entirely left to the SMT solver. Even better, if the available SMT solver (e.g., Yices [7]) offers some support for λ-abstractions, the β-reduction needed to obtain (3) can be delegated to the SMT solver. To the best of our knowledge, this simplicity in the computation of the pre-image is in sharp contrast to current approaches to symbolic model checking of *infinite* state systems available in the literature where computationally expensive operations are required to obtain some normal form that can then be exploited by safety and fix-point computations. We avoid this by directly using first-order formulae and then exploiting the flexibility and scalability of the SMT solver to internalize formulae in appropriate data structures that support efficient satisfiability checks to which both safety and fix-point tests can be reduced. This is similar in spirit to what is current practice in *finite* state model checking, where the BDD package abstracts away the details of the efficient handling of finite sets and related operations on them.

4 Light-weight reachability

Having found suitable hypotheses under which conditions (R1), (R2), and (R3) are satisfied, it is now possible to introduce the algorithm in Figure 1 to compute $BR^n(\tau, K)$ for the class of array-based systems considered in this paper. The function Pre computes the pre-image of an $\exists^I$-formula (according to (1)) and then applies the syntactic manipulations explained in the proof of Proposition 3.5 to find a logically equivalent $\exists^I$-formula. The algorithm in Figure 1 semi-decides the (invariant) model-checking problem for $K(a)$ whenever $K(a)$ is an $\exists^I$-formula. Termination of

BReach for some important classes of systems may be obtained as shown in [22].

4.1 Reducing quantifier instantiation

As observed after Theorem 3.4, checking the A_I^E-satisfiability of $\exists^{A,I}\forall^I$-formulae is possible by integrating quantifier instantiation and SMT solving. Unfortunately, the instantiation is too expensive. One of the main reasons for this is the growing number (at each iteration of the loop in Figure 1) of existentially quantified variables in the prefix of the $\exists^I$-formulae resulting from the computation of the Pre-image: the existentially quantified prefix $\exists\, \underline{k}$ is augmented with $\exists\, \underline{i}$ in (3) (cf. proof of Proposition 3.5). Hence, it would be desirable to find ways to greatly limit the growing number of existentially quantified variables in the prefix of the Pre-image or, even better, to ensure it remains constant. In the rest of this Section, we develop our ideas to achieve this goal so as to obtain a *light-weight* (but still complete) version of the algorithm in Figure 1. To this end, for simplicity, **we make a stronger assumption on the theory T_I of indexes, namely that Σ_I is relational**.

Let $\tau(a, a')$ be the disjunction of the T-formulae

$$\exists \underline{i}\; (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j. F^h(\underline{i}, a[\underline{i}], j, a[j])), \tag{4}$$

where h ranges over a certain finite set S, say $S = \{1, \dots, s\}$. For the sake of simplicity, we assume that the tuple $\underline{i}$ is independent of h: the length of such a tuple is denoted by $c(\tau)$ and it is called the *complexity* of τ.

A formula $K(a)$ *has degree* less than n (in symbols, $d(K) \leq n$) iff K is A_I^E-equivalent to a formula of the form $\exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ in which the length of the tuple $\underline{k}$ is less than or equal to n. When writing $d(K) = n$, we mean that n is the smallest natural number such that $d(K) \leq n$ holds. Now, the proof of Proposition 3.5 shows that the degree of $Pre(\tau, K)$ can be bounded by the sum of the complexity of τ and of the degree of K, in symbols $d(Pre(\tau, K)) \leq c(\tau) + d(K)$. By induction, we derive $d(BR^n(\tau, K)) \leq d(K) + n \cdot c(\tau)$. Below, we show that, under suitable hypotheses, this estimate can be slightly improved.

An *activity condition* is a quantifier-free Σ-formula $\gamma(s, a[s])$ such that

$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \rightarrow \gamma(t, a[t])$ holds for each $h \in S$ and each variable $t \in \underline{i}$.

Discharging this obligation implies that only active processes can fire transition h. An $\exists^I$-formula $\exists \underline{k}\, \psi(\underline{k}, a[\underline{k}])$ is γ-*active* (for the activity condition γ) iff

$A_I^E \models \psi(\underline{k}, a[\underline{k}]) \rightarrow \gamma(t, a[t])$ holds for each variable t in $\underline{k}$.

At this point, it is interesting to consider our formalization of parametrised systems in the stopping failures model. Recall, from Section 2, that no transition is enabled when control reaches the additional state q_{crash}. This suggests $a[s] \neq q_{crash}$ as an obvious candidate for expressing an activity condition in such systems.

Example 4.1 To show that $a[s] \neq \mathtt{crash}$ is an activity condition for our running example, it is necessary to prove the A_I^E-unsatisfiability of the three formulae $a[z] = \mathtt{x} \wedge a[z] = \mathtt{crash}$ where $\mathtt{x} \in \{\mathtt{idle}, \mathtt{wait}, \mathtt{crit}\}$. This is immediate since $\mathtt{crash} \neq \mathtt{x}$ for $\mathtt{x} \in \{\mathtt{idle}, \mathtt{wait}, \mathtt{crit}\}$ by the theory T_I of enumerated data types. The $\exists^I$-formula $K(a)$ in Example 3.3 is γ-active. To see this, it is sufficient to prove that $z_1 \neq z_2 \wedge a[z_1] = \mathtt{crit} \wedge a[z_2] = \mathtt{crit} \wedge a[z_i] = \mathtt{crash}$ are A_I^E-unsatisfiable (for $i = 1, 2$), which is trivial. □

Recall that τ is a disjunction of T-formulae of the form (4), for $h \in S$. We say that τ is γ-*local* iff the formula

$$\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \gamma(s, F^h(\underline{i}, a[\underline{i}], s, a[s])) \to s \in \underline{i} \vee a[s] = F^h(\underline{i}, a[\underline{i}], s, a[s]). \tag{5}$$

is A_I^E-valid for each $h \in S$, where $s \in \underline{i}$ abbreviates $\bigvee_{u \in \underline{i}} s = u$. To understand (5), observe that, once the transition fires, the state of the system is updated according to the assignment $a'[s] := F^h(\underline{i}, a[\underline{i}], s, a[s])$; hence, (5) means that the value stored at an index s of the array a not causing the transition to fire remains the same, unless s becomes 'inactive' after the transition, i.e. unless $\gamma(s, a'[s])$ becomes false (just read (5) by contraposition).

Example 4.2 It is not difficult to see that the transition of the running example is γ-local, where γ is $a[s] \neq \texttt{crash}$ (as in Example 4.1). For the sake of conciseness, we illustrate this only for τ_3 (τ_1 and τ_2 are similar, only more cases must be considered). It is sufficient to check for A_I^E-unsatisfiability the two formulae obtained by case-splitting on (5), namely $a[z] = \texttt{crit} \wedge s = z \wedge \texttt{crash} \neq \texttt{idle} \wedge s \neq z \wedge a[s] \neq \texttt{idle}$ and $a[z] = \texttt{crit} \wedge s \neq z \wedge \texttt{crash} \neq a[s] \wedge s \neq z \wedge a[s] \neq a[s]$. Both checks are trivial. □

In practice, it is possible to find activity conditions making transitions local for several protocols ensuring mutual exclusion as well as for algorithms manipulating arrays (e.g., sorting) by guessing appropriate γ's. Typical examples of non local transitions are those of broadcast protocols.

We are now ready to show the usefulness of local transitions to limit the growing prefix of $\exists^I$-formulae computed by the algorithm in Figure 1.

Theorem 4.3 *Suppose $c(\tau) \geq 1$. Let K be an $\exists^I$-formula and let γ be an activity condition such that K is γ-active and τ is γ-local. Then, $d(BR^n(\tau, K)) \leq d(K) + n \cdot c(\tau) - n$. Hence, if $c(\tau) = 1$ then $d(BR^n(\tau, K)) \leq d(K)$, for all $n \geq 0$.*

Before applying Theorem 4.3, let us consider the task of finding a suitable activity condition γ. For parametrised systems formalized in the stopping failures model, we have seen before Example 4.1 that the obvious candidate for an activity condition is $a[s] \neq q_{crash}$, because no transition is enabled in the additional crash state. In general, the search space for such $\gamma(s, a[s])$ is infinite, but it becomes *finite* for instance when T_E has a quasi-relational signature: in that case, the hypotheses of Theorem 4.3 can be effectively checked, as the following example shows.

Example 4.4 The signature Σ_E of Example 3.2 is quasi-relational, hence we can compute all possible choices for $\gamma(s, a[s])$: the latter can only be a Boolean combinations of atoms of the form $a[z] = x$ for $x \in \{\texttt{idle}, \texttt{wait}, \texttt{crit}, \texttt{crash}\}$. By enumerating such formulae (e.g., in disjunctive normal form) and checking for the conditions of γ-activity and γ-locality, we quickly find that $a[s] \neq \texttt{crash}$ satisfies the desired requirements. □

Case $c(\tau) = 1$. In this case, Theorem 4.3 implies that the number of existentially quantified variables of the pre-image remains constant at each iteration of the loop of the backward reachability algorithm in Figure 1. So, if the input formula K of the reachability algorithm has $\underline{k}$ existentially quantified variables, $BR^i(\tau, K)$ is A_I^E-equivalent to an $\exists^I$-formula of the form $\exists \underline{k} \phi_i(\underline{k}, a[\underline{k}])$ and the A_I^E-validity of $BR^{i+1}(\tau, K) \to BR^i(\tau, K)$, to detect a fix-point, is equivalent to the A_I^E-unsatisfiability of the quantifier-free formula

$$\phi_{i+1}(\underline{k}, a[\underline{k}]) \wedge \bigwedge_{\sigma} \neg \phi_i(\underline{k}\sigma, a[\underline{k}\sigma]),$$

where σ ranges over all possible substitutions with domain $\underline{k}$ and co-domain $\underline{k}$, according to the instantiation procedure sketched after Theorem 3.4. Although the number of instances (or, equivalently, of substitutions σ's) to be considered at each iteration of the loop does not change, it is tempting to furtherly simplify the formula above by considering only one instance, obtained by the identical substitution: $\phi_{i+1}(\underline{k}, a[\underline{k}]) \wedge \neg\phi_i(\underline{k}, a[\underline{k}])$. This algorithm is computationally much less expensive than that suggested by Theorem 3.4; unfortunately, it is incomplete in general. However, when, e.g., T_E is an enumerated datatype theory, checking the A_I^E-unsatisfiability of $\phi_{i+1}(\underline{k}, a[\underline{k}]) \wedge \neg\phi_i(\underline{k}, a[\underline{k}])$ is precise enough, since there are only finitely many quantifier-free formulae of the form $\psi(\underline{k}, a[\underline{k}])$, up to A_I^E-equivalence, and a fix-point can always be reached (maybe with more iterations than those needed by the loop in Figure 1). Operationally, this observation can be implemented by preliminarily 'grounding the whole system,' as exemplified below.

Example 4.5 For the formulae in Example 3.3, we have $c(\tau) = 1$ and $d(BR^n(\tau, K)) = d(K) = 2$ by Theorem 4.3 with the activity condition in Example 4.1. Because T_E is an enumerated datatype theory (cf. Example 3.2) and the last observation above, to prove the safety of the running example, it is sufficient to consider a parametrized system consisting of only $d(K) = 2$ processes, i.e. it is sufficient to consider the following ground version of the system: $\widehat{I}(a) := (a[z_1] = \texttt{idle} \wedge a[z_2] = \texttt{idle})$,

$$\widehat{\tau}(a, a') := \bigvee_{l=1}^{2} (\bigvee_{i=1}^{3} (\phi_L^i(z_l, a[z_l]) \wedge \bigwedge_{m=1}^{2} a'[z_m] = F^i(z_l, a[z_l], z_m, a[z_m]))),$$

and $\widehat{K}(a) := z_1 \neq z_2 \wedge a[z_1] = \texttt{crit} \wedge a[z_2] = \texttt{crit}$, where z_1, z_2 are INDEX constants. It is a routine exercise to verify that the formulae for checking fix-point and safety computed with $\widehat{I}$, $\widehat{\tau}$, and $\widehat{K}$ are the same (modulo trivial logical manipulations) as those obtained by using I, τ, K and then performing the above instantiation. □

Case $c(\tau) = 1\frac{1}{2}$. In practice (see, e.g., the Szymanski protocol [10]), it turns out that parametrised systems with transitions of complexity 2 are formalized by disjunctions of T-formulae of the form

(6) $\exists i_1, i_2\, (\phi_L^h(i_1, i_2, a[i_1], a[i_2]) \wedge a' = \lambda k.F^h(i_1, a[i_1], k, a[k]))$,

i.e. whereas both existentially quantified variables occur in the local component, just one of them occurs in the update. The degree-reducing algorithm of Theorem 4.3 prescribes that, when computing $Pre(\tau, \exists \underline{k}\, \phi)$, one can insert the extra information that one of the i_1, i_2 is equal to one of the $\underline{k}$'s. However, when τ is a disjunction of T-formulae of the form (6), one can improve again the procedure by imposing the condition that *precisely* i_1 must be identified with one of the $\underline{k}$'s. Since this reduces by one half the length of the optimized $Pre(\tau, \exists \underline{k}\phi)$, we (informally) say that formulae (6) have complexity $1\frac{1}{2}$. For the formal details, see the Appendix.

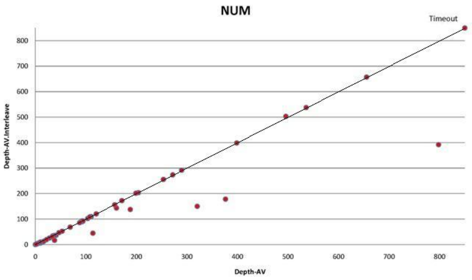
4.2 Refinements of backward reachability and experiments

Theorem 4.3 and its applications suggest to implement the algorithm in Figure 1 on top of a client-server architecture, where the client is a "light-weight" program to generate formulae representing (iterated) pre-images, whose A_I^E-unsatisfiability is checked by an off-the-shelf SMT solver, the server. Below, we discuss how to make this efficient. We assume the available SMT solver to offer the following interface functionalities: (I1) parsing of strings for processing symbolic expressions, (I2) supporting definable function symbols (as an alternative, one may require to support λ-abstraction), and (I3) incremental handling of a logical context, i.e. addition/removal of logical facts and (incremental) satisfiability checks.

Lazy generation of proof obligations. Although (I1) seems sufficient to mechanize our approach as SMT solvers have a standard input format [24], preliminary experiments have shown that formulae for both safety and fix-point checks quickly become quite large and parsing may become a bottleneck. To see this, consider the sequence of formulae generated by the loop of the algorithm in Figure 1: $BR^{i+1}(\tau, K) := BR^i(\tau, K) \vee \mathsf{Pre}(\tau, K^i)$, for $i \geq 0$. The formulae for safety and fix-point checks involving $BR^{i+1}(\tau, K)$ contains a copy of the previously generated (and already parsed by the SMT solver) formula $BR^i(\tau, K)$. After some iteration, parsing becomes prohibitively expensive. To avoid this, we introduce a new Boolean variable BR^i to be used as an "abbreviation" for the arbitrarily complex formula $BR^i(\tau, K)$ in the computation of $BR^{i+1}(\tau, K)$ as follows: $\mathsf{BR}^i \leftrightarrow BR^i(\tau, K)$, which is is added to the logical context of the SMT solver by invoking (I3), so that $BR^{i+1}(\tau, K) := \mathsf{BR}^i \vee \mathsf{Pre}(\tau, K^i)$, for $i \geq 0$. In this way, the size of $BR^{i+1}(\tau, K)$ as well as of all the formulae containing it remains constant over the iterations and parsing is no more problematic.

Interleaving. By definition, our transition formula τ is the disjunction of the T-formulae τ^h and $\mathsf{Pre}(\tau, K)$ is the disjunction of the $\mathsf{Pre}(\tau^h, K)$'s. In practice, it is rarely the case that each $\mathsf{Pre}(\tau^h, K)$ is A_I^E-satisfiable as not all transitions may be taken from a given state. This suggests to check first for the A_I^E-satisfiability of the formula $\mathsf{Pre}(\tau^h, K^i)$: if the result is unsatisfiable, then we proceed to consider τ^{h+1} (if $h+1 \leq s$). In other words, we replace the check for safety with the following sequence of (simpler) checks: ($\mathsf{C1}.h$) $\mathsf{Pre}(\tau^h, K^i) \wedge I$ is A_I^E-satisfiable and the fix-point check with ($\mathsf{C2}.h$) $\neg(\mathsf{Pre}(\tau^h, K^i) \to \mathsf{BR}^i)$ is A_I^E-satisfiable, for $h = 1, ..., s$ at the i-iteration of the loop in the algorithm of Figure 1. If one of the checks ($\mathsf{C1}.h$) is satisfiable, we stop and report the unsafety of the system. Instead, if all the checks ($\mathsf{C2}.h$) are unsatisfiable, we conclude that $\mathsf{Pre}(\tau, K^i) \to \mathsf{BR}^i$ is A_I^E-valid and, hence, a fix-point has been reached. Otherwise, if some of the checks ($\mathsf{C2}.h$) are satisfiable and the others are unsatisfiable, we discard the latter ones and take the disjunction of the former ones to compute $BR^{i+1}(\tau, K)$. By interleaving in this way the generation of the proof obligations and the satisfiability checks, the hope is to generate a more compact symbolic representation of the set of reachable state.

Breadth vs. depth. The algorithm in Figure 1 implements a breadth-first visit of the set of backward reachable states. However, thanks to the flexibility of our declarative approach, it is easy to implement a recursive algorithm implementing a depth-first visit of the state space. Consider the s-ary tree built by labelling its root with K and its s-sons with $K \vee \mathsf{Pre}(\tau^h, K)$ for $h = 1, ..., s$, and recursively repeating this construction. A standard depth-first visit of this tree yields a depth-first visit of the state space. Indeed, the tree is constructed on-the-fly while it is visited by using "local" checks for fix-point and safety, similar to those of the algorithm in Figure 1. The main advantage of the depth-first algorithm is that more compact formulae are generated for the SMT solver. Its main drawback is that it may take much longer to terminate (or even diverge). Fortunately, it is possible to alleviate this problem by storing the set of "already-visited" states, i.e. those states describing a "local" fix-point, in a global variable AV, which is then used in subsequent fix-point checks, as follows: prove the A_I^E-validity of $BR_{df}^{i+1} \to (BR_{df}^i \vee AV)$ for $h \geq 0$, where BR_{df}^{i+1} and BR_{df}^i are the sets of states reachable in depth-first at iteration $i+1$ and i, respectively, and AV is the set of "already-visited" states (at the beginning, AV is *false*, i.e. the empty set of states). When $h = s$, this enhanced depth-first algorithm



The number of variables n_v and the number n_t of T-formulae for problems in NUM are such that $3 \leq n_v \leq 44$ and $3 \leq n_t \leq 37$.
Experiments were conducted on a Pentium Dual-Core 1.66 GHz with 1 Gb Sdram running Linux. All the timings are in seconds and the time-out is 3 hours.
A dot below the diagonal means a better performance of depth-AV.Interleave over depth-AV; viceversa for a dot above.

Fig. 2. Results of SMTMC on NUM

performs a "global" fix-point check similar to that of the algorithm in Figure 1. Our flexible framework allows us to experiment with "hybrid" strategies combining depth- and breadth-first searches.

The tool. To test the practical viability of the client-server architecture designed above and to evaluate the impact of the various heuristics, we implemented SMTMC, a prototype tool which uses Yices 1.0.11 as the SMT solver (in particular, its API lite that supports (I1)–(I3) above) and writing around 1390 source lines of C code.

Our benchmark set consists of problems taken from the distribution of various model-checking tools for infinite state systems, such as Babylon [3], Brain [4], Action Language Verifier [1], ARMC [2], and PFS [5]. We have considered two classes of problems: NUM (with 34 problems), where T_I is an enumerated data type theory and T_E is the theory of Linear Real/Integer Arithmetic; and AIE (with 19 problems), where T_I is the theory of finite and linearly ordered sets and T_E is the theory of an enumerated data type sometimes combined with Linear Integer Arithmetic. Array-based systems in NUM model situations where a fixed and known number of integer variables is updated by the transition systems; e.g., those obtained by counting abstraction [18]. For problems in this class, we have $c(\tau) = 0$, i.e. problems are quantifier-free. The class AIE features (truly parametric) systems with a fixed (either known or unknown) number of elements; e.g., those in [10]. Although SMTMC has been designed for very expressive extensions (covering all problems that can be modeled by generic array-based systems), it is still under major development and its current version, due to insufficient quantifier instantiation, can only handle, in an incomplete way, most of the problems in AIE. On the other hand, actual performances are encouraging and seem comparable with dedicated state-of-the-art tools for problems in the class NUM. Incomplete runs seem to predict the possibility to obtain good results also for problems in AIE. An executable of our tool, the benchmark problems, and the details about the experiments are at [6].

Heuristics. Our experiments have clearly shown that straightforward implementations of breadth- and depth-first search (even in combination with the interleaving of the generation of proof obligations and satisfiability checking) scale up poorly. The more promising results have been obtained with two extensions of depth-first search: depth-AV, where the fix-point check is enhanced by the checks with the "already-visited" set of states, and depth-AV.Interleave, which is similar to depth-AV except for the fact that the generation of proof obligations is interleaved with satisfiability checking according to (C1.h) and (C2.h). Figure 2 shows that the two heuristics are, in practice, equivalent on the problems of NUM as both are capable of solving 87% of the benchmark problems in the given time-out.

5 Discussion and related work

We have presented a refinement of the SMT-based model-checking of array-based systems of [22] that allows us to directly leverage existing SMT solvers by a light-weight implementation effort. The idea of using arrays to represent system states is not new in model-checking (see in particular [26,25]); what seems to be new in our approach is the fully declarative characterization of *both* the topology and the (local) data structures of systems by using theories. This has two advantages. First, implementations of our approach can handle a wide range of topologies without modifying the underlying data structures representing sets of states. This is in contrast with recently available techniques [10,9] for the uniform verification of parametrized systems, which consist of exploring the state space of a system by using a finitary representation of (infinite) sets of states and require substantial modifications in the computation of the pre-image to adapt to different topologies. Second, since SMT solvers are capable of handling several theories in combinations, we can avoid encoding everything in one theory, which has already been proved detrimental to performances in [14,13,1]. SMT techniques were already employed in model-checking [16,11], but only in the bounded case (whose aim is mostly limited at finding bugs, not at full verification).

Babylon [3] is a tool for the verification of counting abstractions of parametrized systems (e.g., multithreaded Java programs [19]). It uses a graph-based data structure to encode disjunctive normal forms of integer arithmetic constraints. Computing pre-images requires computationally expensive normalization, which is not needed for us as SMT solvers efficiently handle arbitrary integer constraints.

Brain [4] is a model-checker for transition systems with finitely many integer variables which uses an incremental version of Hilbert's bases to efficiently perform entailment/satisfiability checking of integer constraints (the results reported in [27] shows that it scales very well). Taking T_I to be an enumerated datatype theory, the array-based systems considered in this paper reduce to those used by Brain.

A recent interesting proposal to uniform verification of parametrized systems is [12], where a decidability result for Σ^0_2-formulae is derived (these are $\exists\forall$-formulae roughly corresponding to those covered by Theorem 3.4 above, for the special case in which the models of the theory T_I are all the finite linear orders). While the representation of states in [12] is (fully) declarative, transitions are not, as a rewriting semantics (with constraints) is employed. Since transitions are not declaratively handled, the task of proving pre-image closure becomes non trivial; e.g., in [12], pre-image closure of Σ^0_2-formulae under transitions encoded by Σ^0_2-formulae ensures the effectiveness of the tests for inductive invariant and bounded reachability analysis, but not for fix-point checks. In our approach, an easy (but orthogonal) pre-image closure result for existential state descriptions (under certain Σ^0_2-formulae representing transitions) gives the effectiveness of fix-point checks, thus allowing implementation of backward search.

Acknowledgements. The 2nd author was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (www.avantssar.eu).

References

[1] Action Language Verifier. http://www.cs.ucsb.edu/~bultan/composite.

[2] ARMC. http://www.mpi-sws.mpg.de/~rybal/armc.

[3] Babylon. http://www.ulb.ac.be/di/ssd/lvbegin/CST.

[4] Brain. http://www.cs.man.ac.uk/~voronkov/BRAIN.

[5] PFS. http://www.it.uu.se/research/docs/fm/apv/tools/pfs.

[6] SMTMC. http://www.thomasvalsecchi.eu.

[7] Yices. http://yices.csl.sri.com.

[8] 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.

[9] P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, number 4590 in LNCS, 2007.

[10] P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Regular model checking without transducers. In *TACAS*, number 4424 in LNCS, 2007.

[11] A. Armando, J. Mantovani, and L. Platania. Bounded Model Checking of Software using SMT Solvers instead of SAT Solvers. In *Proc. of SPIN'06*, number 3925 in LNCS, pages 146–162, 2006.

[12] A. Bouajjani, P. Habermehl, Y. Yurski, and M. Sighireanu. Rewriting systems with data. In *Proc. of Symp. on Fund. of Comp. Th. (FCT 07)*, pages 1–22, 2007.

[13] T. Bultan, R. Gerber, and C. League. Composite model-checking: verification with type-specific symbolic representations. *ACM Trans. on Soft. Eng. an Meth.*, 9(1):3–50, 2000.

[14] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Trans. on Progr. Lang. and Sys.*, 21(4):747–789, 1999.

[15] L. de Moura and N. Bjørner. Efficient e-matching for smt solvers. In *Proc. of CADE*, LNCS, 2007.

[16] L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Proc. CADE*, volume 2392 of *LNCS*, 2002.

[17] D. Déharbe and S. Ranise. Satisfiability solving for software verification. *Int. Journal on STTT*, 2008. To appear.

[18] G. Delzanno. Automatic verification of parameterized cache coherence protocols. In *Proc. of CAV*, number 1855 in LNCS, 2000.

[19] G. Delzanno, J.-F. Raskin, and L. Van Begin. Towards the automated verification of multi-threaded java programs. In *8th Int. Conf. on TACAS*, number 2280 in LNCS, 2002.

[20] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York-London, 1972.

[21] Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In *Proc. of CADE-21*, LNCS, 2007.

[22] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *Proc. of IJCAR*, LNCS, 2008. Extended version available at http://homes.dsi.unimi.it/~ghilardi/allegati/GhiNiRaZu-RI318-08.pdf.

[23] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kauffman Publishers, Inc., 2000.

[24] S. Ranise and C. Tinelli. The SMT-LIB Standard: Version 1.2. Technical report, Dep. of Comp. Science, Iowa, 2006. Available at http://www.SMT-LIB.org/papers.

[25] A. W. Roscoe, R. S. Lazic, and T. C. Newcomb. On model checking data-independent systems with arrays without reset. *Theory and Practice of Logic Programming*, pages 659–693, 2004.

[26] A. W. Roscoe, R. S. Lazic, and Tom Newcomb. On model checking data-independent systems with arrays with whole-array operations. In *Communicating Sequential Processes*. Springer LNCS, 2005.

[27] T. Rybina and A. Voronkov. Using canonical representations of solutions to speed up infinite-state model checking. In *Proc. of CAV*, number 2404 in LNCS, 2002.

[28] T. Rybina and A. Voronkov. A logical reconstruction of reachability. In *Proc. of PSI*, pages 222–237. LNCS 2890, 2003.

[29] R. Sebastiani. Lazy satisfiability modulo theories. *Jour. on Sat., Boolean Modeling and Comp.*, 3:141–224, 2007.

A Appendix

We provide the details to formally prove the key results of our paper. First (Section A.1), we give some lemmas that allows us to derive Theorem 4.3. Second (Section A.2), we introduce the concept of *strong amalgamability* that allows us to show the usefulness of the notion of "$1\frac{1}{2}$ complexity."

A.1 Proof of Theorem 4.3

Recall that a formula $K(a)$ *has degree* less than n (in symbols, $d(K) \leq n$) iff K is A_I^E-equivalent to a formula of the form $\exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ where the length of the tuple $\underline{k}$ is less than or equal to n. We also write that $d(K) = n$ to mean that n is the smallest natural number such that $d(K) \leq n$ holds.[6] The following Lemma states an important property of the notion of degree.

Lemma A.1 *Let K_1, K_2 be $\exists^I$-formulae. Then, there exists an $\exists^I$-formula K such that*

(i) $A_I^E \models K \leftrightarrow (K_1 \vee K_2)$;

(ii) *if $d(K_1) \leq n$ and $d(K_2) \leq n$, then $d(K) \leq n$.*

Proof. Assume that K_1 is A_I^E-equivalent to $\exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ and that K_2 is A_I^E-equivalent to $\exists \underline{j}\, \psi(\underline{j}, a[\underline{j}])$. Up to (bounded) variable renaming, we can assume, w.l.o.g., that $\underline{k}$ is an initial sub-sequence of $\underline{j}$, i.e. $\underline{j} = \underline{k}, \underline{l}$. Hence, we can take K to be $\exists \underline{k}\, \exists \underline{l} (\phi(\underline{k}, a[\underline{k}]) \vee \psi(\underline{k}, \underline{l}, a[\underline{k}], a[\underline{l}]))$ and (i)–(ii) are trivially verified. □

Recall that the transition $\tau(a, a')$ is assumed to be a disjunction of T-formulae of the form (4), i.e.

$$\exists \underline{i}\ (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j. F^h(\underline{i}, a[\underline{i}], j, a[j])),$$

where h ranges over a finite set S. According to the hypothesis of Theorem 4.3, we have $c(\tau) \geq 1$ and, hence, the tuple $\underline{i}$ occurring in (4) is non-empty. Let us also assume that $\gamma(s, a[s])$ is an activity condition such that τ is γ-local.

According to the proof of Proposition 3.5, if $K(a) := \exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$, then $Pre(\tau, K)$ is the formula obtained from

$$\exists \underline{i}\, \exists \underline{k} \bigvee_h \exists a' \, [\phi_L^h(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j. F^h(\underline{i}, a[\underline{i}], j, a[j]) \wedge \phi(\underline{k}, a'[\underline{k}])]$$

by eliminating the quantifier $\exists a'$ and then applying β-conversion; if we write $\phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])]$ for the formula obtained by replacing the occurrences of the terms $a'[\underline{k}_1], \ldots a'[\underline{k}_l]$ by $F^h(\underline{i}, a[\underline{i}], \underline{k}_1, a[\underline{k}_1]), \ldots, F^h(\underline{i}, a[\underline{i}], \underline{k}_l, a[\underline{k}_l])$, respectively, in $\phi(\underline{k}, a'[\underline{k}])$ (here l is the length of the tuple $\underline{k}$), the $\exists^I$-formula which is equivalent to $Pre(\tau, K)$ can be written as

$$\exists \underline{i}\, \exists \underline{k} \bigvee_h \, [\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))]. \qquad (3)$$

[6] This formal definition does not imply that d is a total function: actually, $d(K)$ is defined only when there exists an $\exists^I$-formula $K'(a)$ which is A_I^E-equivalent to K. We choose the above formulation because it makes the notions $d(K) \leq n$ and $d(K) = n$ structural, i.e. invariant under A_I^E-equivalence.

Lemma A.2 *If the $\exists^I$-formula K is γ-active, so is the $\exists^I$-formula (3).*

Proof. Let K be $\exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ and let l be the length of the tuple $\underline{k}$; for every h, we must show that

$$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \rightarrow \gamma(t, a[t]) \tag{A.1}$$

for every $t \in \underline{k} \cup \underline{i}$. This is trivial for $t \in \underline{i}$, because by the definition of activity condition for γ we have

$$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \rightarrow \gamma(t, a[t]) \tag{A.2}$$

for $t \in \underline{i}$. We only need to prove (A.1) for $t \in \underline{k}$. To this end, let us examine the definition of γ-activity for K; such condition guarantees that

$$A_I^E \models \forall a(\phi(\underline{k}, a[\underline{k}]) \rightarrow \gamma(t, a[t])) \tag{A.3}$$

for all $t \in \underline{k}$; by instantiating the universally quantified variable a with $\lambda s. F^h(\underline{i}, a[\underline{i}], s, a[s])$, we get

$$A_I^E \models \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \rightarrow \gamma(t, F^h(\underline{i}, a[\underline{i}], t, a[t])) \tag{A.4}$$

from (A.3), for each t in $\underline{k}$. Thus, the antecedent of (A.1) implies, modulo A_I^E, the formula $\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \gamma(t, F^h(\underline{i}, a[\underline{i}], t, a[t]))$ for all $t \in \underline{k}$. By locality of τ, we then get that the antecedent of (A.1) implies $t \in \underline{i} \vee F^h(\underline{i}, a[\underline{i}], t, a[t]) = a[t]$, [7] i.e.

$$\begin{gathered} A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \rightarrow \\ t \in \underline{i} \vee F^h(\underline{i}, a[\underline{i}], t, a[t]) = a[t] \end{gathered} \tag{A.5}$$

for each t in $\underline{k}$. This implies (A.1) as soon as we take into account (A.2) and (A.4). □

For an $\exists^I$-formula K of the form $\exists \underline{k} \phi(\underline{k}, a[\underline{k}]))$, we let $Pre_*(\tau, K)$ be the formula

$$\exists \underline{i}\, \exists \underline{k}\, \bigvee_h [\bigvee_t (t \in \underline{i}) \wedge \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))], \tag{A.6}$$

where the index t is among those in $\underline{k}$.

Lemma A.3 *We have that $d(Pre_*(\tau, K)) \leq c(\tau) + d(K) - 1$.*

Proof. If $d(K) = 0$, the formula (A.6) contains an empty disjunct (which is—by definition—a contradiction), hence the whole formula (A.6) is a contradiction, has degree 0, and the claim is trivial. [8] Suppose now that $d(K) \geq 1$; distribute the existential quantifiers over disjunctions, eliminate one of the quantified variables in $\underline{k}$ in each disjunct and finally apply Lemma A.1. □

The following Lemma is trivial, by Lemma A.2, because the quantifier-free matrix of (A.6) implies that of (3):

Lemma A.4 *If the $\exists^I$-formula K is γ-active, so is the $\exists^I$-formula $Pre_*(\tau, K)$.*

Lemma A.5 *If K is γ-active, then*

$$K \vee Pre(\tau, K) \text{ is } A_I^E\text{-equivalent to } K \vee Pre_*(\tau, K).$$

---

[7] Recall that we use $t \in \underline{i}$ as an abbreviation for the formula $t = i_1 \vee \cdots \vee t = i_m$, if $\underline{i}$ is the tuple $i_1, \ldots, i_m$.

[8] Recall that $c(\tau) \geq 1$, so that $c(\tau) + d(K) - 1$ is non-negative.

Proof. Since the entailment $A_I^E \models Pre_*(\tau, K) \to Pre(\tau, K)$ is trivial, we need only to show that $A_I^E \models Pre(\tau, K) \to K \vee Pre_*(\tau, K)$. By the same argument employed in Lemma A.2, from the γ-activity of K, we get again (A.5) for all h, namely that

$$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \to \bigwedge_t (t \in \underline{i} \vee F^h(\underline{i}, a[\underline{i}], t, a[t]) = a[t])$$

(here the index t ranges over the tuple of variables $\underline{k}$); as a consequence, also

$$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \to (\bigvee_t t \in \underline{i}) \vee \bigwedge_t F^h(\underline{i}, a[\underline{i}], t, a[t]) = a[t]$$

holds. By basic equational reasoning, this implies that

$$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \to (\bigvee_t t \in \underline{i}) \vee \phi(\underline{k}, a[\underline{k}])$$

and that

$$A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \to (\bigvee_t t \in \underline{i}) \vee K$$

which shows that

$$\begin{aligned} &A_I^E \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])) \to K \vee \\ &\vee [(\bigvee_t t \in \underline{i}) \wedge \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))] \end{aligned}$$

from which the claim follows. □

Proposition A.6 *For an $\exists^I$-formula K, let $BR_*^n(\tau, K)$ be $\bigvee_{s=0}^n Pre_*^s(\tau, K)$;*[9] *if K is γ-active, then $BR^n(\tau, K)$ is A_I^E-equivalent to $BR_*^n(\tau, K)$, for all n.*

Proof. We prove the lemma by induction on n: the case $n = 0$ is trivial, so suppose that $n = m + 1$. Notice that, since $Pre(\tau, -)$ commutes with disjunctions, it is easily seen that $BR^{m+1}(\tau, K)$ is logically equivalent to $K \vee Pre(\tau, BR^m(\tau, K))$. Thus, by induction, we obtain that $BR^{m+1}(\tau, K)$ is A_I^E-equivalent to $K \vee Pre(\tau, BR_*^m(\tau, K))$, i.e. to

$$\begin{aligned} &K \vee Pre(\tau, K) \vee Pre(\tau, Pre_*(\tau, K)) \vee Pre(\tau, Pre_*^2(\tau, K)) \vee \cdots \\ &\cdots \vee Pre(\tau, Pre_*^m(\tau, K)). \end{aligned} \tag{A.7}$$

By Lemma A.4, the $\exists^I$-formulae $Pre_*^s(\tau, K)$ are all γ-active; hence, by Lemma A.5, we can replace $Pre(\tau, K)$ with $Pre_*(\tau, K)$, $Pre(\tau, Pre_*(\tau, K))$ with $Pre_*^2(\tau, K)$, etc. in (A.7) until we obtain $\bigvee_{s=0}^{m+1} Pre_*^s(\tau, K)$, which is $BR_*^{m+1}(\tau, K)$ by definition. □

Now, we are ready to prove our main result.

[9] Obviously, Pre_*^s is the s-th iteration of Pre_*: formally, we have $Pre_*^0(\tau, K) = K$ and $Pre_*^{m+1}(\tau, K) = Pre_*(\tau, Pre_*^m(\tau, K))$.

Proof of Theorem 4.3. By Proposition A.6, to estimate the degree of $BR^n(\tau, K)$ it is sufficient to estimate the degree of $BR^n_*(\tau, K)$. To do this, we first show, by induction on n, that

$$d(Pre^n_*(\tau, K)) \leq d(K) + n \cdot c(\tau) - n. \tag{A.8}$$

For $n = 0$, the claim is trivial. Suppose $n = m + 1$; since we know by Lemma A.3, that $d(Pre_*(\tau, Pre^m_*(\tau, K))) \leq c(\tau) + d(Pre^m_*(\tau, K)) - 1$, by induction, we derive

$$\begin{aligned} d(Pre^{m+1}_*(\tau, K)) &\leq c(\tau) + d(K) + m \cdot c(\tau) - m - 1 = \\ &= d(K) + (m + 1) \cdot c(\tau) - (m + 1), \end{aligned}$$

as desired.

Applying (A.8) and Lemma A.1, we finally get that $d(BR^n_*(\tau, K)))$ is bounded by $d(K) + n \cdot c(\tau) - n$. $\square$

A.2 Complexity $1\frac{1}{2}$

When considering parametrized systems, the complexity of the corresponding transition formulae is either 1 or 2 in practically all cases. The case of complexity 1 has already been discussed above; here, we consider a transition formula τ of complexity 2, where $\tau := \bigvee_{h \in S} \tau_h$ and each τ_h is a T-formula of the form (4). In this case, we have that $Pre_*(\tau, K)$ is

$$\exists \underline{i}\ \exists \underline{k}\ \bigvee_h [\bigvee_t (t = i_1 \vee t = i_2) \wedge \phi^h_L(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))], \tag{A.9}$$

where $\underline{i} = i_1, i_2$ and t is in $\underline{k}$.

Although $c(\tau) = 2$, it may be the case that for some h in S, only i_1 occurs in τ_h whereas i_2 does not. As a consequence, Lemma A.5 (whose proof can be carried out for each $h \in S$, separately) implies that $Pre_*(\tau_h, K)$ (i.e., the corresponding disjunct in $Pre_*(\tau, K)$) can be simplified to

$$\bigvee_t t = i_1 \wedge \phi^h_L(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])), \tag{A.10}$$

where the variables in $\underline{i}, \underline{k}$ are existentially quantified. Although this observation already allows us to simplify $Pre_*(\tau, K)$ (to one half of its original length), there are still those T-formulae where both i_1 and i_2 occur, i.e. those τ_h's which are such that $c(\tau_h) = 2$. Fortunately, a closer look at the form of the τ_h's which are obtained when considering parametrized systems (e.g., the Szkymanski protocol) reveals that while i_1 occurs both in the local and the update component, i_2 occurs only in the local component and it does not occur in the update component. In other words, the τ_h's have the following special form:

$$\exists \underline{i}\ (\phi^h_L(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F^h(i_1, a[i_1], j, a[j])). \tag{A.11}$$

From now on, we consider array-based systems whose transition τ is a disjunction of T-formulae of the form (A.11). Although $c(\tau) = 2$, it turns out that the pre-images for this class of array-based systems can be simplified in a way in-between (A.9) and (A.10) under (mild) additional assumptions. Motivated by this, we introduce *the notion of "complexity of a transition τ equal to $1\frac{1}{2}$"* (in symbols, by abuse of notation, we write $c(\tau) = 1\frac{1}{2}$), to suggest that one of the two existentially quantified variables in $\underline{i}$ occurs with a limited scope.

Two remarks are in order. First, translating the systems in [10] into the formalism of this paper yields T-formulae of the form (A.11) only. Second, examples of array-based systems whose transition τ is such that $c(\tau) = 2$ and $c(\tau) \neq 1\frac{1}{2}$ (i.e. both existentially quantified variables can occur in the T-formulae of τ with unlimited scope) come from the formalization of sorting algorithms (see [22] for details).

When τ has complexity $1\frac{1}{2}$, one cannot avoid the prefix of existential quantifiers to grow during backward search; however, as anticipated above, we can improve on Proposition A.6 as follows. Let τ have complexity $1\frac{1}{2}$ and let K be the $\exists^I$-formula $\exists \underline{k} \phi(\underline{k}, a[\underline{k}]))$; define $Pre_+(\tau, K)$ to be the formula

$$\exists \underline{i}\, \exists \underline{k} \bigvee_h [\bigvee_t t = i_1 \wedge \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F^h(i_1, a[i_1], \underline{k}, a[\underline{k}]))], \tag{A.12}$$

where $\underline{i} = i_1, i_2$ and the index t ranges on the tuple $\underline{k}$. Lemmas A.3 and A.4 obviously apply also to Pre_+; we show that under a mild additional model-theoretic hypothesis, called *strong amalgamation* (to be formally introduced below) on T_I, also Proposition A.6 applies.

Proposition A.7 *Let τ have complexity $1\frac{1}{2}$ and T_I have the strong amalgamation property. For an $\exists^I$-formula K, let $BR^n_+(\tau, K)$ be $\bigvee_{s=0}^n Pre^s_+(\tau, K)$ (here Pre^s_+ is the obvious s-th iteration of Pre_+); if K is γ-active, then $BR^n(\tau, K)$ is A_I^E-equivalent to $BR^n_+(\tau, K)$, for all n.*

Proof. The proof follows step by step the proof of Proposition A.6 using the stronger characterization (A.13) of γ-locality supplied by Lemma A.9 below. □

A.3 Strong amalgamation

Strong amalgamation is a notion arising in algebra and model theory. In our context, we say that a theory T_I has the *strong amalgamation property* if the following condition holds:

> Let $\mathcal{B}_1, \mathcal{B}_2$ be models of T_I having the property that the intersection of their supports is the support of a common Σ_I-substructure which is also a model of T_I (call it $\mathcal{A}$); then, there exists a further model $\mathcal{C}$ of T_I such that $\mathcal{B}_1, \mathcal{B}_2$ are both Σ_I-substructures of $\mathcal{C}$ [10] ($\mathcal{C}$ is called the amalgam of $\mathcal{B}_1, \mathcal{B}_2$ over $\mathcal{A}$).

For many theories, the amalgam $\mathcal{C}$ can be simply obtained by the set-theoretic union of $\mathcal{B}_1$ and $\mathcal{B}_2$. In this way, one can show that T_I has strong amalgamation when the models of T_I are all (finite) sets, all (finite) graphs, etc. For other theories (e.g., when T_I has as models all (finite) linear orders, all (finite) partial orders, all (finite) forests, etc), showing strong amalgamation requires to manipulate a little bit the set theoretic union of two of their models.

The following two Lemmas explain why strong amalgamation is important in the proof Proposition A.7 above.

[10] Unlike standard amalgamation, strong amalgamation is formulated in terms of substructures (and not in terms of embeddings): notice that we insisted on the fact that the intersections of the supports of $\mathcal{B}_1, \mathcal{B}_2$ must be support of $\mathcal{A}$ and that $\mathcal{C}$ is a superstructure of $\mathcal{B}_1, \mathcal{B}_2$. It is possible to formulate strong amalgamation in terms of embeddings, but in that case we must add explicitly the condition that $\mathcal{A}$ is the pullback (i.e. the "intersection") of $\mathcal{B}_1$ and $\mathcal{B}_2$ over $\mathcal{C}$.

Lemma A.8 *Let T_I have the strong amalgamation property. If a quantifier-free $(\Sigma \cup \Sigma_D)$-formula $\theta(\underline{e}, \underline{i}, a[\underline{i}], s, a[s]) \wedge \neg(s \in \underline{i})$ is A_I^E-satisfiable, so is the formula*

$$\theta(\underline{e}, \underline{i}, a[\underline{i}], s, a[s]) \wedge \theta(\underline{e}, \underline{i}, a[\underline{i}], \tilde{s}, a[\tilde{s}]) \wedge \neg(s \in \underline{i}) \wedge \neg(\tilde{s} \in \underline{i}) \wedge s \neq \tilde{s}.$$

Proof. W.l.o.g, assume that θ is a conjunction of $\Sigma_I \cup \Sigma_E$-literals (defined symbols of Σ_D can be eliminated):[11] thus we have that θ can be written as $\theta_I(\underline{i}, s) \wedge \theta_E(\underline{e}, a[\underline{i}], a[s])$, where θ_I, θ_E are conjunctions of Σ_I, Σ_E-literals, respectively. Let $\mathcal{M}$ be a model of $\theta \wedge \neg(s \in \underline{i})$ and let $\mathcal{M}_I, \mathcal{M}_E$ be its reducts to Σ_I, Σ_E. Let $\mathcal{N}_I$ be the Σ_I substructure of $\mathcal{M}_I$ generated by the $\underline{i}$[12] (this is still a model of T_I because T_I is closed under substructures) and let $\mathcal{M}'_I$ be an isomorphic copy of $\mathcal{M}_I$ in which only the elements which are not in the support of $\mathcal{N}_I$ have been renamed. Notice that $\mathcal{M}'_I$ is a model of $\theta_I(\underline{i}, \tilde{s}) \wedge \neg(\tilde{s} \in \underline{i})$ if we assign to $\tilde{s}$ the renamed copy of (the element previously assigned to) s. Let us strongly amalgamate $\mathcal{M}_I$ and $\mathcal{M}'_I$ over $\mathcal{N}_I$: since the amalgam is strong[13] and since truth of quantifier-free formulae extends to super-structures, the amalgam (let us call it $\mathcal{A}_I$) satisfies $\theta_I(\underline{i}, s) \wedge \theta_I(\underline{i}, \tilde{s}) \wedge s \neq \tilde{s} \wedge \neg(s \in \underline{i}) \wedge \neg(\tilde{s} \in \underline{i})$. The pair $\mathcal{A}_I, \mathcal{M}_E$ gives rise to a model of A_I^E larger than the original $\mathcal{M}$ (as far as the index support is concerned). We extend the array (previously assigned to) a to an array defined on the domain of $\mathcal{A}_I$ by letting $a[j] := a[s]$ for every j which is not in the support of $\mathcal{M}_I$. Under this extension, we have that also $\theta_E(\underline{e}, a[\underline{i}], a[s]) \wedge \theta_E(\underline{e}, a[\underline{i}], a[\tilde{s}])$ is true (because $a[s] = a[\tilde{s}]$ and $\theta_E(\underline{e}, a[\underline{i}], a[s])$ was already true in $\mathcal{M}$). $\square$

Lemma A.9 *Let T_I have the strong amalgamation property. Furthermore, let τ have complexity $1\frac{1}{2}$ and be γ-local. Then the formula*

(A.13) $\quad \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \gamma(s, F^h(i_1, a[i_1], s, a[s])) \rightarrow s = i_1 \vee a[s] = F^h(i_1, a[i_1], s, a[s]).$

is A_I^E-valid for each h (here, again, $\underline{i} := i_1, i_2$).

Proof. Suppose it is not the case. Then, there is a model $\mathcal{M}$ of A_I^E satisfying

$$\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \gamma(s, F^h(i_1, a[i_1], s, a[s])) \wedge s \neq i_1 \wedge a[s] \neq F^h(i_1, a[i_1], s, a[s]).$$

By γ-locality, we must have that $\mathcal{M} \models s = i_2$ (under the current assignment to the variables i_1, i_2, s, a), hence

$$\mathcal{M} \models \phi_L^h(\underline{i}, a[\underline{i}]) \wedge \gamma(i_2, F^h(i_1, a[i_1], i_2, a[i_2])) \wedge i_2 \neq i_1 \wedge a[i_2] \neq F^h(i_1, a[i_1], i_2, a[i_2]).$$

By Lemma A.8, we can 'duplicate i_2 over i_1', thus getting a model of the following formulae

$$\phi_L^h(i_1, i_2, a[i_1], a[i_2]),\ \gamma(i_2, F^h(i_1, a[i_1], i_2, a[i_2])),\ i_2 \neq i_1,\ a[i_2] \neq F^h(i_1, a[i_1], i_2, a[i_2]),$$

$$\phi_L^h(i_1, \tilde{i}_2, a[i_1], a[\tilde{i}_2]),\ \gamma(\tilde{i}_2, F^h(i_1, a[i_1], \tilde{i}_2, a[\tilde{i}_2])),\ \tilde{i}_2 \neq i_1,\ a[\tilde{i}_2] \neq F^h(i_1, a[i_1], \tilde{i}_2, a[\tilde{i}_2])$$

[11] To eliminate a defined symbol of the kind $G(\underline{i}, s, a[\underline{i}], a[s], \underline{e})$, replace its occurrences by a new variable d of sort `ELEM` and add the defining formula equivalent to $G(\underline{i}, s, a[\underline{i}], a[s], \underline{e}) = d$ to θ.

[12] We identify the variables $\underline{i}$ with the elements of the support of $\mathcal{M}_I$ assigned to them by the assignment satisfying $\theta \wedge \neg(s \in \underline{i})$ (such identifications will be systematic in the following).

[13] This means that, according to the above definition of strong amalgamation property, the intersections of the supports of $\mathcal{M}_I$ and $\mathcal{M}'_I$ must be precisely the support of $\mathcal{N}_I$.

in which $i_2 \neq \tilde{i}_2$ also holds. But then the γ-locality condition for τ is contradicted (take $s := \tilde{i}_2$). $\square$

Some final observations are in order. As mentioned above, Proposition A.7 applies whenever a stronger form of γ-locality holds, namely condition (A.13). This condition can be checked directly (its shape falls within our decidability results, so the check is effective), however in concrete examples there is no need at all of such a check, because condition (A.13) follows from ordinary γ-locality. For the latter to be true, however, some condition seems to be needed; we just proved that strong amalgamation on T_I is a sufficient condition.