A Uniform Deductive Approach for Parameterized Protocol Safety

Jean-François Couchot, Alain Giorgetti, and Nikolai Kosmatov
LIFC, University of Franche-Comté
25030 Besançon, France
{couchot,giorgett,kosmatov}@lifc.univ-fcomte.fr

ABSTRACT

We present a uniform verification method of safety properties for classes of parameterized protocols. Properties like mutual exclusion or cache coherence are automatically verified for any number of similar processes communicating by broadcast and rendez-vous. The protocols are specified in a language of generalized substitutions on array data structures. Sets of states are expressed by first-order formulae with equality. Predecessors are computed by an iterative semi-algorithm. Reaching an initial state or the fixpoint is shown to be decidable and an original decision procedure is provided. As a running example, the MESI protocol illustrates this approach. Experimental results show its applicability to various properties and protocol classes.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification.

General Terms: Verification.

Keywords: symbolic model checking, assertion, reachability, safety, generalized substitutions.

1. INTRODUCTION

A protocol is a piece of code executed in parallel on many processes communicating with each other to perform a global functionality like mutual exclusion, leader election or cache coherence. Such a protocol is *parameterized* [1] when it is the same for any number of cooperating processes. The challenge is to verify its global effect uniformly, i.e. once for all its sizes.

Decidability results for the parametric verification of safety properties only concern restricted classes of parameterized protocols [4, 3]. The more pragmatic approach of *rich-language symbolic model checking* [8, 7, 5, 9] reduces this decidability question to the termination of a fixpoint computation on an adequate abstraction grouping states in sets, with procedures deciding the inclusion of sets of states and the emptiness of their intersection. Along this line we study the case where the parameter ranges over a finite set without any special structure. Consequently, the global system can be modelled by arrays indexed by this set. Our main contributions are a proof that this abstract point of view is adequate to symbolic model checking, a description of many classes of protocols falling in this case, and an original implementation based on a powerful theorem prover.

ASE'05, November 7–11, 2005, Long Beach, California, USA.

2. PARAMETERIZED PROTOCOLS

An operational model for a protocol is a labeled transition system (S, L, T), where S is a common finite set of process states, L is a set of action labels and $T \subseteq S \times L \times S$ is a set of labeled transitions. Moreover, this transition system is parameterized by the identifier i of the process p_i executing the protocol, in the sense that transitions can also be guarded by conditions on process identifiers (e.g. i and j with $j \neq i$) and on the current state of some other processes.

We distinguish three kinds of action. A *local action* changes the state of a single process. It is denoted by a label $l \in L$. A *rendez-vous* is a synchronization between two processes. One process sends a message according to an output transition $(s, l!, s') \in T$ and another one receives it by moving along an input transition $(r, l?, r') \in T$. A *broadcast* action changes all the process states. A single process sends a broadcast message to all the processes along a transition $(s, l!!, s') \in T$, whereas each other process in some state t moves to some state t' such that $(t, l??, t') \in T$. For sake of clarity, this study is restricted to deterministic broadcast reception: for each state $t \in S$ and each broadcast label $l??$, there exists exactly one state t' such that $(t, l??, t') \in T$.

This operational model includes the classes of broadcast protocols [4] and of client-server protocols [3]. From the algorithmic point of view, it can model mutual exclusion and cache coherence protocols, among others.

As a running example, we use the MESI *cache coherence* protocol [5] which ensures that each process has access to the same memory location. Its transition system is represented in Figure 1. Initially, all the processes have an invalid cache (I state). Properties of interest are detailed in Section 3.

3. MODELLING LANGUAGE

This section defines a language of generalized substitutions suitable to model the global behavior of any number of communicating processes executing a protocol under consideration. This language is built upon a many-sorted first-order language of predicates. We define data types, expressions, first-order formulae and the syntax of indeterministic substitutions in this order.

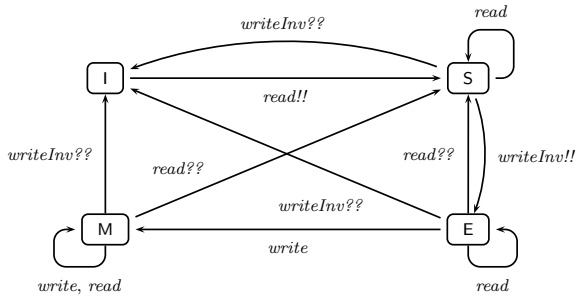

Figure 1: MESI transition system.

$$
\begin{array}{rcl}
expr & ::= & arr \mid ele \mid ind \\
arr & ::= & a \mid \mathsf{z} \mid \mathsf{wr}(arr, ind, ele) \mid \mathsf{const}(ele) \\
 & & \mid \mathsf{block}(arr, \{(\mathsf{e}_1, ele), \ldots (\mathsf{e}_n, ele)\}) \\
ele & ::= & \mathsf{e} \mid \mathsf{rd}(arr, ind) \\
ind & ::= & i \mid \mathsf{j} \\
subst & ::= & \mathsf{skip} \mid assign \mid pred \Longrightarrow subst \mid (@j \ . \ subst) \\
assign & ::= & x := expr \mid x := expr \mid\mid assign \\
pred & ::= & litt \mid pred \wedge pred \mid pred \vee pred \\
 & & \mid (quant \ j \ . \ pred) \\
litt & ::= & equa \mid \neg equa \\
equa & ::= & ele = ele \mid ind = ind \\
quant & ::= & \forall \mid \exists
\end{array}
$$

where $a \in V_{arr}$, $\mathsf{z} \in \mathsf{C}_{arr}$, $\mathsf{e} \in \mathsf{C}_{ele}$, $i \in V_{ind}$ and $\mathsf{j} \in \mathsf{C}_{ind}$.

Figure 2: Language grammar.

A global configuration is defined by the states of all the processes. It is stored in an array of sort *arr* taking values in a set of protocol states abstracted away by the *ele* sort and indexed by a (finite non empty) set of processes abstracted away by the *ind* sort. Thus the processes become anonymous and can be distinguished only by a name considered as a constant within a fixed set denoted by C_{ind}.

Expressions are generated by the *expr* syntactic element of Figure 2. This grammar is parameterized by three finite disjoint sets C_{arr}, C_{ele} and C_{ind} of constants of sort *arr*, *ele* et *ind* respectively, and by two disjoint sets V_{arr} and V_{ind} of variables of sort *arr* and *ind* respectively. Constants are written in the sans-serif font and variables in *italic*. Suppose that $\mathrm{card}(\mathsf{C}_{ele}) = n$ and $\mathsf{C}_{ele} = \{\mathsf{e}_1, \mathsf{e}_2, \ldots, \mathsf{e}_n\}$.

Intuitively, given a term a of sort *arr*, a term i of sort *ind* and a term e of sort *ele*, the term $\mathsf{rd}(a, i)$ stands for the element of the array a at the index i. The term $\mathsf{wr}(a, i, e)$ stands for the array obtained from a by setting the value at the index i to e. The term $\mathsf{const}(e)$ denotes the constant array, whose value at every index is the same element $e \in ele$. The array $\mathsf{block}(a, \{(\mathsf{e}_1, e'_1), \ldots, (\mathsf{e}_n, e'_n)\})$ is obtained from a by replacing each value e_l by e'_l $(1 \leq l \leq n)$. All these definitions are formalized by the set $\mathcal{A}$ of axioms of Figure 3, where all the variables are universally quantified and sorted, namely i and j of sort *ind*, e and e'_i $(1 \leq i \leq n)$ of sort *ele* and a of sort *arr*. To avoid the quantification on l $(1 \leq l \leq n)$, (4) must be repeated n times, once for each value of l.

$$
\begin{array}{ll}
i \neq j \Rightarrow \mathsf{rd}(\mathsf{wr}(a, i, e), j) = \mathsf{rd}(a, j) & (1) \\
\mathsf{rd}(\mathsf{wr}(a, i, e), i) = e & (2) \\
\mathsf{rd}(\mathsf{const}(e), i) = e & (3) \\
\mathsf{rd}(a, i) = \mathsf{e}_l \Rightarrow & \\
\quad \mathsf{rd}(\mathsf{block}(a, \{(\mathsf{e}_1, e'_1), \ldots (\mathsf{e}_n, e'_n)\}), i) = e'_l & (4)
\end{array}
$$

Figure 3: Axiom set $\mathcal{A}$ for the extended array theory.

$$
\begin{array}{rl}
s_{write} & =_{\mathsf{def}} (@j \,.\, \mathsf{rd}(a, j) = \mathsf{e} \Longrightarrow s := \mathsf{wr}(a, j, \mathsf{m})) \\
s_{inv} & =_{\mathsf{def}} (@j \,.\, \mathsf{rd}(a, j) = \mathsf{s} \Longrightarrow s := \mathsf{wr}(\mathsf{const}(\mathsf{i}), j, \mathsf{e})) \\
s_{read} & =_{\mathsf{def}} (@j \,.\, \mathsf{rd}(a, j) = \mathsf{i} \Longrightarrow \\
 & \quad s := \mathsf{wr}(\mathsf{block}(a, \{(\mathsf{s}, \mathsf{s}), (\mathsf{e}, \mathsf{s}), (\mathsf{m}, \mathsf{s}), (\mathsf{i}, \mathsf{i})\}), j, \mathsf{s}))
\end{array}
$$

Figure 4: MESI substitutions.

The operational part of our language is defined by generalized substitutions along the *subst* syntactic element of Figure 2. In Figure 2 and all that follows, $x \in V_{arr} \cup V_{ind} \cup V_{ele}$ is a variable of any sort, whereas $j \in V_{ind}$ is a variable of sort *ind* only. Each kind of substitution specifies a command. Basic substitutions are skip which leaves all the variables unchanged, and simple assignment (:=) which changes a single variable value. All the assignments are assumed well-sorted. Simultaneous assignment of two variables and more can be done with the associative and commutative multiple assignment operator ||. Substitutions can be guarded ($\Longrightarrow$) by a predicate (*pred*). The predicate language is the fragment of the first-order logic with equality and sorts *arr*, *ind* and *ele* where only the variables of sort *ind* can be quantified. The propositional connectives $\Rightarrow$ and $\Leftrightarrow$ can be introduced as shortcuts. The indeterminism required to model the interleaved behavior of processes is modelled by the unbounded choice binder @, limited, like quantifiers, to variables of sort *ind*. This restriction is of importance for the existence of a decision procedure (see Section 5). Consequently, the syntax of quantifiers ($\forall$, $\exists$) and of the choice binder @ does not mention the sort of the bounded variables.

As we shall see, this language is sufficient to model the protocols under consideration. For example, Figure 4 shows a model of the MESI protocol. The substitutions of Figure 4 use a single variable $a \in V_{arr}$ representing the global configuration whose domain is the set of processes hidden behind the *ind* sort. The fact that a process p_j is in a state $t \in \mathsf{C}_{ele} = \{\mathsf{m}, \mathsf{e}, \mathsf{s}, \mathsf{i}\}$ is represented by $\mathsf{rd}(a, j) = t$.

An *assertion* is a predicate representing a set of states. In the following, we consider the assertion language of all the predicates defined by the syntactic element *pred* in Figure 2.

A safety property says that, in given circumstances, the system should not evolve towards critical or error states. When the context is reduced to a set of initial states I, such a property says that some set of states E should not be reached by the system starting from some state in I. Then, this reachability question is modelled by a pair (I, E) of sets of states, or, equivalently, by a pair of assertions (*Source*, *Target*).

The set of initial states for the MESI example is defined by the assertion $Source =_{\mathsf{def}} (\forall j \,.\mathsf{rd}(a, j) = \mathsf{i})$ which means that all caches are initially in the I state. Two mutual exclu-

$$
\begin{array}{rcl}
\langle P \Longrightarrow S\rangle C & = & P \wedge \langle S\rangle C \\
\langle \mathsf{skip}\rangle C & = & C \\
\langle x := E\rangle C & = & C(E/x) \\
\langle y_1 := E_1 \mid\mid \cdots \mid\mid y_n := E_n\rangle C & = & C(z_2/y_2)\ldots(z_n/y_n) \\
 & & (E_1/y_1)(E_2/z_2)\ldots(E_n/z_n) \\
\langle (@j\ .\ S)\rangle C & = & (\exists j\ .\ \langle S\rangle C)
\end{array}
$$

where $P \in \mathit{pred}$, $S \in \mathit{subst}$, x is a variable, $C(E/x)$ denotes the syntactic replacement in C of all the free occurrences of x by E, $y_1, \ldots y_n$ are pairwise distinct, and $z_2, \ldots z_n$ are pairwise distinct fresh variables.

Figure 5: $\langle\,\rangle$ calculus.

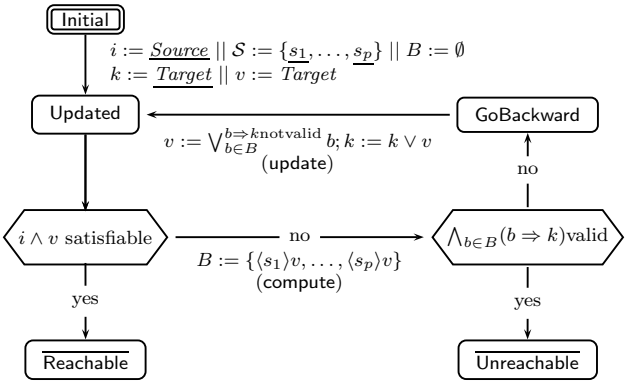

Figure 6: Backward reachability (semi-)algorithm.

sion properties can be considered. (i) Two processes cannot simultaneously write into the memory, i.e. the assertion

$$
\begin{array}{l}
\mathit{Target}_{\mathit{write}} =_{\mathsf{def}} (\exists j_1\ .\ (\exists j_2\ . \\
\qquad j_1 \neq j_2 \wedge \mathsf{rd}(a, j_1) = \mathsf{m} \wedge \mathsf{rd}(a, j_2) = \mathsf{m}))
\end{array} \tag{5}
$$

is not reachable. (ii) A process cannot share the memory for reading it while another process is modifying it. Formally,

$$
\begin{array}{l}
\mathit{Target}_{\mathit{read}} =_{\mathsf{def}} (\exists j_1\ .\ (\exists j_2\ . \\
\qquad j_1 \neq j_2 \wedge \mathsf{rd}(a, j_1) = \mathsf{s} \wedge \mathsf{rd}(a, j_2) = \mathsf{m}))
\end{array} \tag{6}
$$

is not reachable.

4. BACKWARD REACHABILITY

As shown in [9], a backward reachability approach associated with an assertion transformer is an easy method to avoid unmanageable quantifiers. In this section, we firstly present a symbolic backward computation step as the syntactic action of a generalized substitution on an assertion. Next, we iterate this calculus in a fixpoint computation.

For an assertion C and a generalized substitution S, $\langle S\rangle\, C$ denotes the assertion characterizing the states that have a successor by S satisfying C. Figure 5 explicitly defines the action of the assertion transformer $\langle S\rangle C$ as a syntactic calculus. Removing quantifiers in assertions is a key of success for their automatical discharging into a prover. An important remark for what follows is that the calculus of $\langle S\rangle C$ can introduce only quantifiers over indexes, for any assertion C and any substitution S from the language of Section 3.

The (semi-)algorithm computing predecessors is given in Figure 6. The input (resp. output) data is underlined (resp. overlined). The semantics of the internal variables is as follows: i is the initial assertion, $\mathcal{S}$ is the set of substitutions, the assertion k characterizes the backward visited states, v defines the predecessors of the v assertion of the previous iteration that are not already in k and B is the set of computed predecessors of v.

At each iteration the semi-algorithm sequentially verifies the satisfiability of $i \wedge v$, computes the set B of predecessor assertions of v for each substitution s_i $(1 \leq i \leq p)$, checks whether $\bigwedge_{b\in B}(b \Rightarrow k)$ is valid and updates the assertions v and k.

Even if there exists a decision procedure for the $i \wedge v$ satisfiability and for the $\bigwedge_{b\in B}(b \Rightarrow k)$ validity, this semi-algorithm can diverge by non termination of the fixpoint calculus. Abstraction or acceleration techniques can sometimes prevent it but are not the object of this study. The decidability of evolution conditions is then discussed.

5. DECIDING FIXPOINT CONDITIONS

This section establishes the decidability of the conditions arising in the semi-algorithm of Section 4. We assume the usual notions of formulae, satisfiability, validity and theories. A formula φ is called *satisfiable modulo a theory* $\mathcal{T}$, or $\mathcal{T}$-*satisfiable,* if $\mathcal{T} \wedge \varphi$ is satisfiable. The conditions to verify can be expressed as the satisfiability of some formulae modulo the many-sorted theory of arrays $\mathcal{T}_{\mathcal{A}}$ defined by the set $\mathcal{A}$ of axioms of Figure 3. The following result can be used for a large class of such formulae encountered in practice.

Theorem 1 *Consider the many-sorted first-order predicate language defined in Section 3 and a closed predicate φ in which no existential quantifier is in the scope of a universal one. Then the satisfiability of φ modulo $\mathcal{T}_{\mathcal{A}}$ is decidable.*

Proof. Construct the Skolem form of φ. The skolemization of φ can only introduce some constants since no existential quantifier in φ is in the scope a universal one. Let ψ be the Skolem form of φ written in prenex form. Hence ψ is a closed Skolem formula of the form $\forall x_1 \ldots \forall x_k\ .\, \Phi(x_1, \ldots, x_k)$ for some variables $x_1, \ldots, x_k$ $(k \geqslant 0)$ of sort *ele* and a quantifier free formula $\Phi(x_1, \ldots, x_k)$. The result follows from Corollary 1 of [6] by induction on k. □

We can now show the decidability of the conditions in the semi-algorithm of Section 4 for the considered properties.

Corollary 1 *Suppose that Source, Target and all guards in the substitutions $s_1, \ldots, s_p$ are closed predicates in which no existential quantifier is in the scope of a universal one. Then the satisfiability of the reachability condition $i \wedge v$ is decidable at every iteration for the semi-algorithm of Figure 6.*

Proof. We see from Figure 5 that during the computation of $\langle s_l\rangle\, v$, new quantifiers can come from v, from @ in s_l or from the guards of s_l. Besides, the only external quantifiers added during the computation of k are existential ones. We see by induction on the execution length that the reachability condition $i \wedge v$ has no existential quantifier under a universal one, so the result follows from Theorem 1. □

Corollary 2 *Suppose that Target and all guards in the substitutions $s_1, \ldots, s_p$ are closed predicates without universal quantifiers. Then the validity of the inclusion conditions $b \Rightarrow k$ is decidable at every iteration for the semi-algorithm of Figure 6.*

Model	Property	Size	Steps	Time
Pidset [2]	Mutual exclusion	4	1	0.7 s.
Dijkstra [1]	Mutual exclusion	4	3	21.3 s.
MESI	Cache coherence	3	3	17.9 s.
S. German [3]	Cache coherence	10	4	29.4 s.

Table 1: Experimental results.

PROOF. The validity of $b \Rightarrow k$ is equivalent to the unsatisfiability of $b \wedge \neg k$. As in the previous proof, we can show by induction that at each iteration, the predicates b ($b \in B$) and k contain no universal quantifiers. The predicate $\neg k$ contains no existential quantifier. Therefore the predicate $b \wedge \neg k$ satisfies the conditions of Theorem 1 and its satisfiability is decidable. □

In the MESI example of Figure 1 the *Source*, $Target_{write}$ and $Target_{read}$ properties and the guards of the substitutions satisfy the conditions of Corollaries 1,2. Therefore the verification of all conditions in the backward reachability semi-algorithm is decidable for the considered safety properties (5) and (6).

6. EXPERIMENTS

The semi-algorithm of Figure 6 is implemented in Java. A pre-process replaces terms containing const or block symbols with predicates built on rd and wr symbols. Then each evolution condition is sent to the haRVey prover[1] to be discharged modulo the equational theory axiomated by axioms (1) and (2) of Figure 3. Table 1 summarizes the experimental results[2] of this study. The second column describes the safety property being checked and the third one gives the number of substitutions in the model. The number of iterations needed to establish the property is given in the fourth column. These and some other examples are detailed at `http://lifc.univ-fcomte.fr/~couchot/specs/`.

7. CONCLUSION

This work addresses the question of the fast detection of classes of parameterized protocols whose safety properties can be verified by symbolic model checking. Before devoting time to look for an ad hoc abstraction or a tailor-made combination of decision procedures, it is suggested to simply translate the global system configurations into arrays and to model data types by first-order axioms. Sets of states can then be represented by assertions in a fragment of a many-sorted first-order logic with equality whose adequacy to symbolic model checking can be stated by general arguments of first-order logic.

This is a basic but original way to check safety properties for many classes of communicating protocols, without dedicated methods. Moreover, it is now proved that reaching an initial state or a fixpoint is decidable within a general theory of array data structures. This decision procedure is based on quantifier expansion and sorts and does not make a claim for efficiency. However, our experiments give good results by using a variant based on the superposition capabilities of the haRVey prover. The decidability of this optimization remains to be proved.

We intend to apply this approach to other classes of algorithms and data structures, combining experimental implementations and theoretic investigations. Experimentations quickly answer the feasability question whereas finding an ad hoc decision procedure is much harder work. When our experimental laboratory shows that this deductive approach is too light to catch a termination argument, or when more efficiency is required, we suggest to look for abstractions, approximations and a clever combination of decision procedures. In the other cases, it is satisfactory to get a symbolic model checking running above an existing theorem prover and to explain its success within the classical theory of first-order logic with sorts and equality.

8. REFERENCES

[1] K. Baukus, Y. Lakhnech, and K. Stahl. Verification of Parameterized Protocols. *Journal of Universal Computer Science*, 7(2):141–158, 2001.

[2] J.-F. Couchot and A. Giorgetti. Analyse d'atteignabilité déductive. In *Congrès Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL'04*, pages 269–283, 2004.

[3] G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In *Proc. of the 7th Int. Conf. on Principles and Practice of Constraint Programming (CP'01)*, volume 2239 of *LNCS*, pages 286–301. Springer, 2001.

[4] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based analysis of broadcast protocols. In *CSL*, pages 50–66, 1999.

[5] G. Delzanno and A. Podelski. Constraint-based deductive model checking. *Int. Journal on Software Tools for Technology Transfer*, 3(3):250–270, 2001.

[6] P. Fontaine and E. P. Gribomont. Decidability of invariant validation for parameterized systems. In *Proc. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 97–112. Springer, 2003.

[7] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Computer Aided Verification, 9th Int. Conf. (CAV'97)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[8] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Computer Aided Verification, 9th Int. Conf. (CAV'97)*, volume 1254 of *LNCS*, pages 424–435. Springer, 1997.

[9] T. Rybina and A. Voronkov. A logical reconstruction of reachability. In *Perspectives of System Informatics*, volume 2890 of *LNCS*, pages 222–237. Springer, 2003.

[1] `www.loria.fr/equipes/cassis/softwares/haRVey/`

[2] run on a Centrino 1.5 Ghz with 512 Mb RAM.