A Uniform Deductive Approach for Parameterized Protocol Safety

Jean-François Couchot
LIFC, Univ. de Franche-Comté
25030 Besançon France
couchot@lifc.univ-fcomte.fr

Alain Giorgetti
LIFC, Univ. de Franche-Comté
25030 Besançon France
giorgett@lifc.univ-fcomte.fr

Nikolai Kosmatov
LIFC, Univ. de Franche-Comté
25030 Besançon France
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 communication protocols are specified in a language of generalized substitutions on array data structures. State configurations 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.

1. INTRODUCTION

A communication protocol is a piece of code executed in parallel on many processes communicating with each other in order to perform a global functionality like mutual exclusion, leader election or cache coherence. Such a protocol is *parameterized* [6] when it is the same for any number of cooperating processes. The challenge is then to verify its global effect uniformly, i.e. once for all its sizes.

Whereas the general question of verifying properties of parameterized protocols is undecidable [3], many recent works [6, 8] have identified restricted classes of parameterized protocols with decidable properties. Works along the line of *rich-language symbolic model-checking* [20] propose an ad hoc combination of abstraction, fixpoint computation and procedures deciding the inclusion of sets of states and the emptiness of their intersection. One direction called *regular model-checking* [1, 20] takes profit of decision procedures on word or tree automata. Another one [5, 9, 12, 21] uses linear arithmetic constraints after a convenient counting abstraction. Both assume there is some link between the parameters and integers. However, for a finite network of processes working in parallel, introducing a linear structure on the processes is not always natural, and counting the number of processes in a given state does not always preserve the properties of interest. For instance, the counting approach prevents to distinguish a process among the others. Predicate abstraction [19] combined with model-checking has been successfully applied on systems where this distinction is needed (e.g. [6]).

We suggest a more basic but unifying approach where the parameter ranges over a finite set without any special structure. As a consequence, the global system is more directly modelled by arrays indexed by this set. Our terminology and point of view owe much to the logical approach of reachability, defined in [25] for guarded assignment systems. We simply add indeterministic actions and remove arithmetic. In this context, our main contributions are a proof that this abstract point of view is nevertheless adequate to symbolic model-checking, a description of many classes of communication protocols falling in this case, and an original implementation based on a powerful tool combining a BDD- or SAT-based propositional layer and an equational prover. Sets of states are represented by assertions in a many-sorted first-order language with equality. The adequacy arguments use two non trivial decidability results. The first one is based on equational reasoning modulo the first-order theory of arrays. The second one is based on an adaptation of Herbrand lemma to a many-sorted context.

The rest of the paper is organized as follows. Section 2 presents classes of protocols supported by this study. Section 3 defines a language of generalized substitutions to model their global behavior and a sublanguage of assertions to model sets of states. The well-known symbolic backward reachability semi-algorithms are revisited in Section 4. Section 5 proves that the satisfiability and entailment checking for the assertions built by these semi-algorithms are decidable. Finally, an implementation and its experimental results are described in Section 6. A running example illustrates the discourse throughout the sections.

2. PARAMETERIZED COMMUNICATION PROTOCOLS

An operational model for a communication 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.

There are three kinds of action labels, for three forms of synchronization between processes. 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. A first process sends a message according to an output transition $(s, l!, s') \in T$ and a second 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 $(s, l!!, s') \in T$. This process moves from the state s to the state s', whereas each other process in some state t moves to some state t' such that $(t, l??, t')$ is in T. For sake of clarity, this study is restricted to deterministic broadcast reception i.e. such that for each state $t \in S$ and each broadcast label $l??$, there exists exactly one state t' such that $(t, l??, t')$ is in T. Consequently, the transitions $(q, l??, q)$ are not shown in figures.

This operational model includes the classes of broadcast protocols [14] and of client-server protocols [13]. From the algorithmic point of view, it can model mutual exclusion protocols [6] and cache coherence protocols [24], among others.

As a running example we use the MESI *cache coherence* protocol [24] which ensures that each process has access to the same memory location. Its transition system is represented in Figure 1. Each process has its own cache and can read it by a local action *read* in any state except I in which the cache is *Invalid*. In the *Shared* state S, cache contents of all the processes are the same and identical to the memory copy. Before modifying its own cache content, a process in the shared state asks the other caches to be invalid by sending them the broadcast message *writeInv*. It enters the state E which means that it owns the *Exclusive* cache allowed to write into the memory and into the cache. After a local writing step, its content is *Modified* (state M). An invalid process (in state I) aiming to read the memory sends the *read* broadcast message. Other processes in the states M and E react to this message by moving into the state S. Initially, all the processes have an invalid cache. We are interested in checking (i) whether the system becomes data inconsistent, i.e. the data being read is not identical to the memory data and (ii) whether two processes write simultaneously into the memory.

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 defined as a many-sorted first-order language of predicates. We start by defining expressions, data types and the corresponding sorts. Then we introduce first-order formulae and finally an operational formalism of indeterministic substitutions.

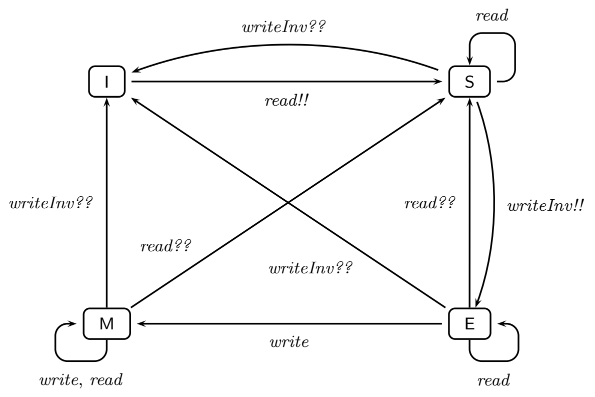


Figure 1: MESI transition system.

$$
\begin{array}{rcl}
\mathit{expr} & ::= & \mathit{arr} \,|\, \mathit{ele} \,|\, \mathit{ind} \;\; ; \\
\mathit{arr} & ::= & \mathsf{wr}(\mathit{arr}, \mathit{ind}, \mathit{ele}) \,|\, \mathsf{const}(\mathit{ele}) \\
 & & |\, \mathsf{block}(\mathit{arr}, \mathit{ele}, \mathit{ele}, \ldots, \mathit{ele}) \\
 & & |\, a \,|\, \mathsf{z} \;\;, \;\; \text{where } a \in V_{arr} \text{ and } \mathsf{z} \in \mathsf{C}_{arr} \;\; ; \\
\mathit{ele} & ::= & \mathsf{rd}(\mathit{arr}, \mathit{ind}) \,|\, \mathsf{e} \;\;, \;\; \text{where } \mathsf{e} \in \mathsf{C}_{ele} \;\; ; \\
\mathit{ind} & ::= & i \,|\, \mathsf{j} \;\;, \;\; \text{where } i \in V_{ind} \text{ and } \mathsf{j} \in \mathsf{C}_{ind}.
\end{array}
$$

Figure 2: Array expressions.

3.1 Expressions

A global configuration is defined by the states of all the processes. It is stored in an array of sort *arr*, indexed by a (finite non empty) set of processes abstracted away by the sort *ind*, and taking values in a set of protocol states abstracted away by the sort *ele*. The sort *ind* abstracts away the number and topology of processes. 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}.

Language expressions are generated by the grammar of Figure 2, 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. The constants will be written in the sans-serif font, and the variables in *italic* as usual. 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 $e \in \mathit{ele}$. In the term $\mathsf{block}(\mathit{arr}, \mathit{ele}, \mathit{ele}, \ldots, \mathit{ele})$, the sort *ele* appears n times. The array $\mathsf{block}(a, e'_1, e'_2, \ldots, e'_n)$ is obtained from a by replacing each value e_l by e'_l. In other words, its value at the index i is e'_l if and only if the value of a at i is e_l.

$$i \neq j \Rightarrow \mathsf{rd}(\mathsf{wr}(a,i,e),j) = \mathsf{rd}(a,j) \quad (1)$$
$$\mathsf{rd}(\mathsf{wr}(a,i,e),i) = e \quad (2)$$
$$\mathsf{rd}(\mathsf{const}(e),i) = e \quad (3)$$
$$\mathsf{rd}(a,i) = \mathsf{e}_l \Rightarrow \mathsf{rd}(\mathsf{block}(a,e'_1,e'_2,\ldots,e'_n),i) = e'_l \quad (4)$$

Figure 3: Extended array theory axioms.

$$\begin{array}{rcl}
\mathit{subst} & ::= & \mathsf{skip} \mid \mathit{assign} \mid \mathit{choice} \mid \mathit{pred} \Longrightarrow \mathit{subst} \\
\mathit{assign} & ::= & x := \mathit{expr} \mid x := \mathit{expr} \mid\mid \mathit{assign} \\
\mathit{choice} & ::= & \mathit{subst} [] \mathit{subst} \mid (@j \ . \ \mathit{subst}) \\
\mathit{pred} & ::= & \mathit{litt} \mid \mathit{pred} \wedge \mathit{pred} \mid \mathit{pred} \vee \mathit{pred} \mid (\mathit{quant}\ j \ . \ \mathit{pred}) \\
\mathit{litt} & ::= & \mathit{equa} \mid \neg \mathit{equa} \\
\mathit{equa} & ::= & \mathit{arr} = \mathit{arr} \mid \mathit{ele} = \mathit{ele} \mid \mathit{ind} = \mathit{ind} \\
\mathit{quant} & ::= & \forall \mid \exists
\end{array}$$

Figure 4: Syntax of predicates and substitutions.

All these definitions are formalized by the axioms of Figure 3, where all the small letters are universally quantified sorted variables, 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 $(1 \leq l \leq n)$, the axiom (4) must be repeated n times, once for each $l = 1, 2, \ldots, n$, what is possible for any given protocol with a fixed number of states n. Since n is independent of the number of processes, this does not restrict our uniform approach. These axioms can be considered as predicates written in the many-sorted first-order language defined in the following section.

For convenience of notation, $\mathsf{block}(a, e'_1, \ldots, e'_n)$ will be also denoted by $\mathsf{block}(a, \{(\mathsf{e}_1, e'_1), \ldots, (\mathsf{e}_n, e'_n)\})$, where the pairs (e_i, e'_i) with the replaced and the new value can be written in arbitrary order.

3.2 Predicates and Operations

The operational part of our language is defined by generalized substitutions along the grammar rules of Figure 4 based on the syntactic elements *expr*, *arr*, *ind* and *el* of Figure 2. In Figure 4 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 only a variable of sort *ind*.

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*).

Predicates are considered in normal negative form [22] (NNF for short) i.e. negations operate only on equalities (*equa*). Thus, each quantifier (∀, ∃) has the same polarity (universal or existential) as in the prenex form of the predicate. This point simplifies the exposure of our proof method in Section 5. 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 ⇒ and ⇔ can be introduced as shortcuts.

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

Figure 5: MESI substitutions.

The indeterminism required to model the interleaved behavior of processes is modelled by two structures. The first one is the associative and commutative bounded choice operator [] between two substitutions. The second one is 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 (∀, ∃) and of the choice binder @ does not mention the sort of the bound variables.

As we shall see, this language is sufficient to model the protocols under consideration. As an example, Figure 5 shows a model of MESI protocol. The substitutions of Figure 5 use a single variable $a \in V_{arr}$ representing the global configuration whose domain is the set of processes hidden behind the sort *ind*. 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 two assertion languages. The language of all the predicates defined by the syntactic element *pred* in Figure 4 and a smaller language of *constraints* defined as predicates without universal quantifiers.

3.3 Safety Properties

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. Consequently, this reachability question is modelled by a pair (I, E) of sets of states, or, equivalently, by a (*Source*, *Target*) pair of assertions.

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

- Two processes cannot simultaneously write into the memory, i.e. the assertion

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

  is not reachable.

- A process cannot share the memory for reading it while

$$
\begin{array}{rcl}
\langle P \Longrightarrow S\rangle\, C & = & P \wedge \langle S\rangle\, C \\
\langle \mathsf{skip}\rangle\, C & = & C \\
\langle x := E\rangle\, C & = & C(E/x) \\
\langle \mathit{assign}\rangle\, C & = & C(z_2/y_2)\ldots(z_n/y_n)(E_1/y_1) \\
 & & \quad (E_2/z_2)\ldots(E_n/z_n) \\
\langle S \,[]\, T\rangle\, C & = & \langle S\rangle\, C \vee \langle T\rangle\, C \\
\langle(@j\ .\ S)\rangle\, C & = & (\exists j\ .\ \langle S\rangle\, C)
\end{array}
$$

Figure 6: $\langle\,\rangle$ calculus.

another process is modifying it. Formally,

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

is not reachable.

4. BACKWARD REACHABILITY

There are basically two kinds of reachability procedures: the backward ones compute predecessors, whereas the forward ones compute successors of a set of states. Section 4.1 presents a symbolic backward computation step as the syntactic action of a generalized substitution on an assertion. Section 4.2 explains why the backward method outperforms the forward one for generalized substitutions. This calculus is iterated in semi-algorithms presented in Section 4.3.

4.1 Computation of Predecessors

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 6 explicitly defines the action of the assertion transformer $\langle S\rangle C$ as a syntactic calculus, derived from the definition [2] of its dual $[S]C$ using the duality relation

$$[S]\, C =_{\mathsf{def}} \neg\langle S\rangle\, \neg C.$$

In this figure, S and T are two generalized substitutions, P is a predicate, x is a variable and the expression $C(E/x)$ denotes the syntactic replacement in C of all the free occurrences of x by E. Moreover, *assign* stands for the parallel assignment $y_1 := E_1 \mid\mid \cdots \mid\mid y_n := E_n$, where the variables $y_1, \ldots, y_n$ are pairwise distinct, and $z_1, \ldots, z_n$ are pairwise distinct fresh variables.

Removing of quantifiers in the assertions is a key of success for their automatical discharging into a prover [7]. 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 next part shows that the backward search has an advantage over the forward one in terms of quantifier handling.

4.2 Predecessors vs Successors

The strongest postcondition predicate transformer $[\,]^o$ [16] can be used to compute the assertion $C[S]^o$ characterizing the successors of C by the substitution S. Its computation

$$
\begin{array}{rcl}
C[P \Longrightarrow S]^o & = & (P \wedge C)[S]^o \\
C[\mathsf{skip}]^o & = & C \\
C[x := E]^o & = & (\exists x'\ .\ C(x'/x) \wedge x = E(x'/x)) \\
C[\mathit{assign}]^o & = & (\exists z_1, \ldots, z_n\ .\ C(z_1/y_1)\ldots(z_n/y_n) \wedge \\
 & & \quad y_1 = E_1(z_1/y_1)\ldots(z_n/y_n) \wedge \cdots \wedge \\
 & & \quad y_n = E_n(z_1/y_1)\ldots(z_n/y_n) \\
C[S \,[]\, T]^o & = & C[S]^o \vee C[T]^o \\
C[(@j\ .\ S)]^o & = & (\exists j\ .\ C[S]^o)
\end{array}
$$

Figure 7: $[\,]^o$ calculus.

rules are presented in Figure 7, with the same conventions as for Figure 6.

Applied on an assertion C with the particular substitution $a := \mathsf{wr}(a, i, e)$, where a is of sort *arr*, i of sort *ind* and e of sort *ele*, the strongest postcondition yields the assertion

$$C[a := \mathsf{wr}(a, i, e)]^o = (\exists a'\ .\ C(a'/a)\ \wedge\ a = \mathsf{wr}(a', i, e)))$$

that contains a quantified variable a' of sort *arr*. This kind of quantification has been carefully excluded of our language since it yields formulae whose satisfiability is generally undecidable.

We see that the backward procedure is more appropriate for reachability analysis in this context than the forward one. This point is to some extent similar to the applicability of Theorem 1 in [5].

4.3 Backward Reachability

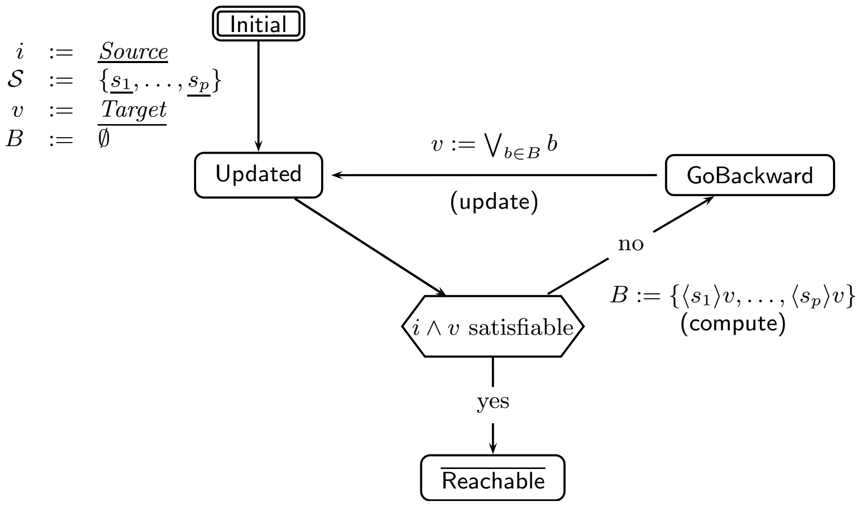
Two (semi)-algorithms computing the predecessors are given in Figures 8 and 9.

In the first one, each rounded rectangle denotes a state and the hexagon denotes an external decision procedure execution. The input (respectively, output) data is underlined (respectively, overlined). By an iteration, we mean a cycle beginning and ending in the state Updated and passing once via compute and update transitions. The semantics of the internal variables is as follows: i is the initial assertion, $\mathcal{S}$ is the set of substitutions, v is the assertion defining predecessors of the value v computed by the previous iteration and B is the set of computed predecessors of v.

At each iteration the semi-algorithm sequentially

1. verifies the satisfiability of $i \wedge v$ (in the positive case the target is reachable),
2. computes the set B of predecessor assertions of v for each substitution s_i $(1 \leq i \leq p)$,
3. updates the assertion v.

Even if there exists a decision procedure for the $i \wedge v$ satisfiability (see part 5.1), this semi-algorithm can diverge for two reasons. The first one is the non termination of the fixpoint calculus, corresponding to the undecidability of the



$i := \underline{Source}$
$\mathcal{S} := \{\underline{s_1}, \dots, \underline{s_p}\}$
$v := \underline{Target}$
$B := \emptyset$
Initial
Updated
GoBackward
$v := \bigvee_{b \in B} b$
(update)
no
$B := \{\langle s_1 \rangle v, \dots, \langle s_p \rangle v\}$
(compute)
$i \wedge v$ satisfiable
yes
$\overline{\text{Reachable}}$

Figure 8: Backward reachability.

invariance checking problem for general parameterized systems [3]. Well funded abstraction [17] or acceleration techniques can sometimes prevent it but are out of the scope of this study. The second one is the unreachability of the target: the algorithm never stops computing assertions even if these one become constant. The next semi-algorithm eliminates the later non termination cause.

The second semi-algorithm (Figure 9) refines the first one by inserting into the (compute) transition a verification of the target unreachability. Compared to the former semi-algorithm, the k assertion characterizes the backward visited states and v defines the predecessors of the v assertion of the previous iteration that are not already into k. This heuristic permits to reduce the proof obligations size of the both termination conditions.

The next section establishes a scope in which the decidability of the semi-algorithms conditions is proved.

5. DECIDING FIXPOINT CONDITIONS

This section studies the decidability of the conditions which must be verified in the semi-algorithms 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. Similarly, a formula φ is called *valid modulo a theory $\mathcal{T}$*, or *$\mathcal{T}$-valid*, if $\mathcal{T} \Rightarrow \varphi$ is valid. The conditions to verify can be expressed as the satisfiability of some formulae modulo the many-sorted theory of arrays defined by the axioms of Figure 3.

We establish the decidability of the satisfiability separately for two classes of formulae. First we show the decidability by superposition for constraints, i.e. predicates without universal quantifiers. Then we propose a new decision procedure for a more general class: formulae in which no existential quantifier is in the scope of some universal one.

5.1 Constraint Case

Recent results providing satisfiability procedures for some theories [4, Section 7] allow us to state the following proposition. We denote by $\mathcal{A}_s$ the many-sorted theory of arrays defined by the axioms (1), (2).

Proposition 1 *If φ is a constraint, then the $\mathcal{A}_s$-satisfiability of φ is decidable by superposition.*

PROOF The first step is skolemization of φ. Let ψ be the obtained Skolem form, it is $\mathcal{A}_s$-satisfiable iff φ is. To decide whether ψ is $\mathcal{A}_s$-satisfiable, construct its disjunctive normal form θ and apply the decision procedure by superposition given in [4, Section 7] to each conjunction of literals in θ. Note that the skolemization of φ did not introduce any new non-constant function symbol since φ does not contain any universal quantifier, so the conditions of [4, Section 7] are satisfied. □

For a theory $\mathcal{T}$ different from $\mathcal{A}_s$, the superposition computation provides a semi-decision procedure which terminates if φ is not $\mathcal{T}$-satisfiable. In case φ is $\mathcal{T}$-satisfiable, the termination is not guaranteed.

Theorem 1 *If C and the guards of the substitution S are all constraints, then $\langle S \rangle\, C$ is also one. Its $\mathcal{A}_s$-satisfiability is decidable by superposition.*

PROOF We see from Figure 6 that during the computation of $\langle S \rangle\, C$, the quantifiers can only come either from @ in S, or from the guards of S, or from C. All these quantifiers are existential ones. It easily follows by induction that $\langle S \rangle\, C$ is a constraint, hence its satisfiability is decidable by Proposition 1. □

Corollary 1 *Suppose that in the semi-algorithms of Figures 8 and 9, Source, Target and the guards of the substitutions $s_1, \dots, s_p$ are constraints. Then the satisfiability of the reachability condition $i \wedge v$ is decidable by superposition at each iteration.*

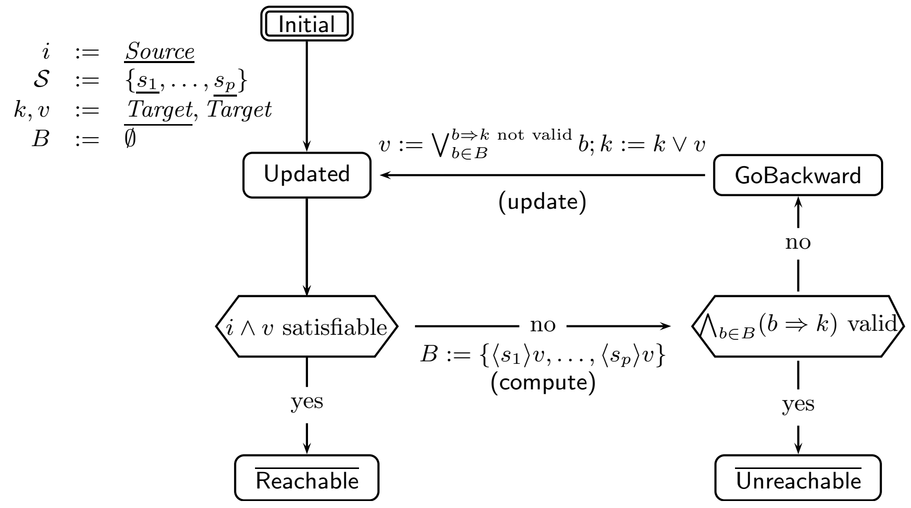

Initial
$i := \underline{Source}$
$\mathcal{S} := \{\underline{s_1}, \dots, \underline{s_p}\}$
$k, v := \underline{Target}, Target$
$B := \emptyset$
Updated
$v := \bigvee_{b \in B}^{b \Rightarrow k \text{ not valid}} b; k := k \vee v$
(update)
GoBackward
no
$i \wedge v$ satisfiable
no
$B := \{\langle s_1 \rangle v, \dots, \langle s_p \rangle v\}$
(compute)
$\bigwedge_{b \in B}(b \Rightarrow k)$ valid
yes
yes
$\overline{\text{Reachable}}$
$\overline{\text{Unreachable}}$

Figure 9: Backward reachability and unreachability.

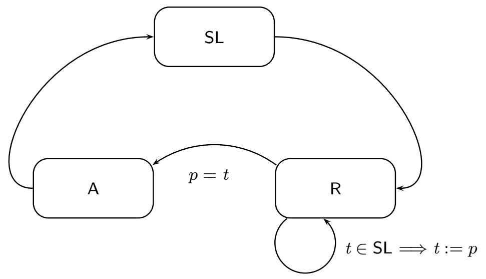


Figure 10: Dijkstra transition system.

Proof The initial values of predicates i and v are constraints. We show by induction on the number of iterations that at each iteration, the new value of v is a constraint, since it is the disjunction of $\langle s_i \rangle\, v$ which are constraints by Theorem 1. Therefore $i \wedge v$ is a constraint at each iteration, so its satisfiability is decidable by superposition. □

The simplified Dijkstra example [6] is presented as an application of this corollary. Each process p controlled by this mutual exclusion algorithm follows the transition system given in Figure 10, where SL, R and A denote the sets of sleeping, ready and active processes, respectively. The corresponding process states are denoted by sl, r and a. The variable t shared between processes indicates which process has the right to be active.

The substitutions corresponding to this system are shown in Figure 11. They are composed only of guarded local actions on the variables $a \in V_{arr}$ and $t \in V_{ind}$.

$$\begin{aligned}
s_{ask} &=_{\mathsf{def}} \quad (@p\ .\mathsf{rd}(a,p) = \mathsf{sl} \Longrightarrow a := \mathsf{wr}(a,p,\mathsf{r}))\\
s_{get} &=_{\mathsf{def}} \quad (@p\ .\mathsf{rd}(a,p) = \mathsf{r} \wedge \mathsf{rd}(a,t) = \mathsf{sl} \Longrightarrow t := p)\\
s_{active} &=_{\mathsf{def}} \quad (@p\ .t = p \wedge \mathsf{rd}(a,p) = \mathsf{r} \Longrightarrow a := \mathsf{wr}(a,p,\mathsf{a}))\\
s_{sleep} &=_{\mathsf{def}} \quad (@p\ .\mathsf{rd}(a,p) = \mathsf{a} \Longrightarrow a := \mathsf{wr}(a,p,\mathsf{sl}))
\end{aligned}$$

Figure 11: Dijkstra substitutions.

In the classical mutual exclusion context, the initial state would be characterized by the assertion

$$\mathit{Source}_{\text{Dijkstra}} =_{\mathsf{def}} (\forall p.\ \mathsf{rd}(a,p) = \mathsf{sl}) \wedge (\exists q.\ t = q)$$

stating that all the processes are initially sleeping and that the variable t is initialized with some index associated to a process. The mutual exclusion property is similar to (5):

$$\begin{gathered}\mathit{Target}_{\text{Dijkstra}} =_{\mathsf{def}} (\exists p_1\ .\ (\exists p_2\ .\\ p_1 \neq p_2 \wedge \mathsf{rd}(a,p_1) = \mathsf{a} \wedge \mathsf{rd}(a,p_2) = \mathsf{a})).\end{gathered}$$

To illustrate the constraint case application, we consider now the assertion

$$\mathit{Source}_3 =_{\mathsf{def}} \mathsf{rd}(a,\mathsf{j}) = \mathsf{sl} \wedge (\exists q\ .\ t = q)$$

where j is a constant. Source_3 states that t is initialized as above and that the process associated to j is sleeping, without providing the status of other processes. To verify whether j and another process can be simultaneously active, consider the assertion

$$\mathit{Target}_3 =_{\mathsf{def}} \mathsf{rd}(s,\mathsf{j}) = \mathsf{a} \wedge (\exists i.\ i \neq \mathsf{j} \wedge \mathsf{rd}(s,i) = \mathsf{a}).$$

The conditions of Corollary 1 are satisfied, so the first semi-algorithm given in Figure 8 permits to check if Target_3 is reachable. Indeed, it detects that Target_3 is reachable from Source_3, so j and another process can be simultaneously active.

Nevertheless, the constraint case is not sufficient to ensure the decidability of the satisfiability verification for all conditions arising from our semi-algorithms. For example, the constraint case does not allow to consider (in both semi-algorithms) the initial assertions *Source* and $Source_{\mathrm{Dijkstra}}$ containing universal quantifiers. For the second semi-algorithm, the decidability for the validity of the inclusion condition $b \Rightarrow k$ is not ensured by the constraint case either. Indeed, if k contains at least one existential quantifier, $b \wedge \neg k$ is not a constraint, and its unsatisfiability (which is equivalent to the validity of $b \Rightarrow k$) is not necessarily detected. Therefore the second semi-algorithm can diverge while calling the superposition prover. The next section establishes the decidability for a larger class of formulae which will allow us to cover all the considered cases.

Moreover, notice that the simplified Dijkstra protocol is a simple example where a counting abstraction is not possible without modifying the transitions: since the number of process states is finite, it is easy to count the processes in each of them. But to express the status of the distinguished process t, it would be necessary to introduce another variable expressing in which state is the process t. Its value would be either sl, or r, or a. Then this value would need to be changed in each operation contrary to the present case where it is only modified in the substitution s_{active}.

5.2 Decision Procedure

Now we give a decision procedure based on the result [18] for a much larger class of predicates. Let us recall the context and a quantifier elimination result of [18].

Let $\mathcal{L} = \langle \mathcal{T}, \mathcal{V}, \mathcal{F}, \mathcal{P}, r, d\rangle$ be a many-sorted first-order language with equality such that $\mathcal{T}$ is a finite set of sorts, $\mathcal{V}$ is the union of disjoint finite sets $\mathcal{V}_\tau$ of variables of sort $\tau \in \mathcal{T}$, $\mathcal{F}$ and $\mathcal{P}$ are sets of function and predicate symbols respectively, $r : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}$ assigns an arity to each function and predicate symbol, $d : \mathcal{F} \cup \mathcal{P} \to \mathcal{T}^*$ defines the signature $d(f) \in \mathcal{T}^{r(f)+1}$ of each function symbol $f \in \mathcal{F}$ and $d(p) \in \mathcal{T}^{r(p)}$ of each predicate symbol $p \in \mathcal{F}$. The Herbrand domain of the sort $\tau \in \mathcal{T}$ is denoted by H_τ and is defined as the set of all well-sorted terms of sort τ (if this set is empty for $\mathcal{L}$, we create a new constant symbol of sort τ in $\mathcal{L}$).

Proposition 2 ([18, Cor. 1]) *Let $\tau \in \mathcal{T}$ be a sort such that*

$$\begin{array}{c}\textit{there is no function symbol } f \textit{ in } \mathcal{L} \\ \textit{of signature } d(f) = (\tau_1\,, \ldots\,, \tau_n\,, \tau),\ n \geqslant 1,\end{array} \qquad (7)$$

and let x be a variable of sort τ. Suppose that $\forall x \,.\, \Phi(x)$ is a closed formula in Skolem form on $\mathcal{L}$. Then $\forall x \,.\, \Phi(x)$ is satisfiable if and only if the finite conjunction $\bigwedge_{c \in H_\tau} \Phi(c)$ is.

We can apply this result to the language introduced in Section 3.

Theorem 2 *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 the theory defined by the axioms of Figure 3 is decidable.*

Proof The considered language is a many-sorted first-order language with equality, where

$\mathcal{T} = \{arr, ele, ind\}$, $\mathcal{F} = \{\mathsf{rd}, \mathsf{wr}, \mathsf{block}, \mathsf{const}\}$,

$d(\mathsf{rd}) = (arr, ind, ele)$, $d(\mathsf{wr}) = (arr, ind, ele, arr)$,

$d(\mathsf{const}) = (ele, arr)$,

$d(\mathsf{block}) = (arr, ele, ele, \ldots, ele, arr)$,

where ele appears $n = \mathrm{card}(\mathsf{C}_{ele})$ times in $d(\mathsf{block})$ (*cf* Section 3.1).

We see that there is no function symbol of arity $\geqslant 2$ ending in the sort ind, so the condition (7) of Proposition 2 for the sort $\tau = ind$ is fulfilled.

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, so the condition (7) of Proposition 2 for the sort $\tau = ind$ is still verified after the skolemization.

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 Proposition 2 by induction on k. $\Box$

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

Corollary 2 *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-algorithms of Figures 8 and 9.*

Proof We see from Figure 6 that during the computation of $\langle s_l\rangle\, v$, new quantifiers can come either from @ in s_l, or from the guards of s_l, or from v. Besides, the only external quantifiers added during the computation of k are existential ones. It is easily seen 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 2. $\Box$

Corollary 3 *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 9.*

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 2 and its satisfiability is decidable. $\Box$

The properties *Source*, $Target_{write}$, $Target_{read}$ and the guards of the substitutions in the MESI example of Figure 1 satisfy

Model	Property	Size	Steps	Time
Pidset	Mutual exclusion	4	1	0.7
Dijkstra	Mutual exclusion	4	3	21.3
Mesi	Cache-coherence	3	3	17.9
S. German	Cache-coherence	10	4	29.4

Table 1: Experimental results.

the conditions of Corollaries 2,3. Therefore the verification of all conditions in the backward reachability semi-algorithm is decidable for the considered safety properties (5) and (6).

6. EXPERIMENTS

From the methodological point of view, backward reachability checking for an assertion and inductive invariant constructing for its negation are equivalent [20]. Our tool thus takes benefit of existing invariant verification tools for the B method. The input data of our tool is a B abstract machine [2] built as follows:

- array structures (that do not exist in B) are represented by total functions with the same domain and codomain.
- The INITIALISATION substitution of the machine corresponds to the *Source* assertion.
- The INVARIANT predicate of the machine is the conjunction of $\neg\,Target$ and the predicates assigning a type to each state variable.

The semi-algorithms of Figures 8 and 9 are implemented in Java. They send each evolution condition to the haRVey[1] [11] validity prover. This prover automatically decides the validity of a ground formulae φ modulo an equational theory $\mathcal{T}$ by refuting the $\mathcal{T}$-satisfiability of $\neg\varphi$. The Skolem form of $\neg\varphi$ is represented by a BDD each branch of which is checked to be satisfiable by a superposition theorem prover.

Table 1 summarizes the experimental results corresponding to this study. The first column gives the name of the protocol under consideration. The second one describes the safety property being checked. The third one gives the size of the model that is the number of substitutions. The number of iterations needed to establish the property is given in the fourth column. The execution time[2] is expressed in seconds in the last column.

Pidset [15] is a mutual exclusion algorithm with synchronization by rendez-vous already presented in [10]. S. German [23, 13] is a cache coherence protocol with many clients and one server. Notice that each example is amenable by a predicate abstraction and finite model-checking method that nevertheless requires the human effort to find the abstraction. Other examples are available at `http://lifc.univ-fcomte.fr/~couchot/specs/` web page.

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

[2] All experiments have been run on a Centrino 1.5 Ghz computer with 512 Mb of RAM under Linux Mandrake 10.1.

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 claim for efficiency. Our experiments use a variant based on the superposition capabilities of the haRVey prover for better results. The decidability of this optimization remains to be proved.

We intend to apply this deductive 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 like the Composite Symbolic Library [26] to find a more specific procedure. In the other cases, it is satisfactory to get a symbolic model-checking running above an existing theorem prover and to dispose of all the theoretical material of first-order logic with sorts and equality to explain its success.

8. REFERENCES

[1] P.-A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In *Computer Aided Verification, 11th International Conference, (CAV'99)*, volume 1633 of *LNCS*, pages 134–145. Springer–Verlag, 1999.

[2] J.-R. Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, 1996.

[3] Krzysztof Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.

[4] A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Journal of Information and computation*, 183:140–164, 2003. Special Issue on the 12th International Conference on Rewriting Techniques and Applications (RTA'01).

[5] Constantinos Bartzis and Tevfik Bultan. Efficient image computation in infinite state model checking. In *Computer Aided Verification, 15th International*

Conference, (CAV'03), volume 2725 of *LNCS*, pages 249–261. Springer–Verlag, 2003.

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

[7] J.-P. Bodeveix and M. Filali. Reduction and quantifier elimination techniques for program validation. *Formal Methods in System Design*, 20(1):69–89, 2002.

[8] Marco Bozzano and Giorgio Delzanno. Beyond parameterized verification. In *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference (TACAS'02)*, volume 2280 of *LNCS*, pages 221–235. Springer–Verlag, 2002.

[9] Tevfik Bultan, Richard Gerber, and William Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, 1999.

[10] 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.

[11] D. Déharbe and S. Ranise. BDD-driven first-order satisfiability procedures (extended version). Rapport de recherche 4630, LORIA, 2002.

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

[13] Giorgio Delzanno and Tevfik Bultan. Constraint-based verification of client-server protocols. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'01)*, volume 2239 of *LNCS*, pages 286–301, 2001.

[14] Giorgio Delzanno, Javier Esparza, and Andreas Podelski. Constraint-based analysis of broadcast protocols. In *CSL*, pages 50–66, 1999.

[15] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Industrial-Strength Formal Methods, First International Symposium of Formal Methods Europe (FME'93)*, volume 670 of *LNCS*, pages 268–284. Springer–Verlag, 1993.

[16] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Springer–Verlag, 1990.

[17] A. Finkel and Ph. Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256:63–92, 2001.

[18] Pascal Fontaine and E. Pascal Gribomont. Decidability of invariant validation for parameterized systems. In *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, (TACAS'03)*, volume 2619 of *LNCS*, pages 97–112. Springer–Verlag, 2003.

[19] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In *Computer Aided Verification, 9th International Conference, (CAV'97)*, volume 1254 of *LNCS*. Springer–Verlag, 1997.

[20] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Computer Aided Verification, 9th International Conference, (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435, 1997.

[21] M. Maidl. A unifying model checking approach for safety properties of parameterized systems. In *Computer Aided Verification, 13th International Conference, (CAV'01)*, volume 2102 of *LNCS*, pages 311–323. Springer–Verlag, 2001.

[22] A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In *Handbook of Automated Reasoning*, chapter 6, pages 335–367. Elsevier, 2001.

[23] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, (TACAS'01)*, volume 2031 of *LNCS*. Springer–Verlag, 2001.

[24] Fong Pong and Michel Dubois. Verification techniques for cache coherence protocols. *ACM Comput. Surv.*, 29(1):82–126, 1997.

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

[26] Tuba Yavuz-Kahveci, Murat Tuncer, and Tevfik Bultan. A library for composite symbolic representations. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, (TACAS'01)*, volume 2031 of *LNCS*, pages 52–66. Springer–Verlag, 2001.