Algorithmic Verification of Invalidation-based Protocols

Marco Bozzano and Giorgio Delzanno

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, via Dodecaneso 35, 16146 Italy
giorgio@disi.unige.it

Abstract. We propose an extension of the model of Broadcast Protocols in which individual processes are allowed to have *unbounded local data* and to communicate via *value passing*. Our specification language is based on *multiset rewriting over first order atomic formulas* enriched with a mechanism for *global synchronization* to model broadcasts, and *constraints* to model the relations over *internal data* and *value passing*. For this new class of parameterized systems, we provide a symbolic validation procedure for checking safety properties, and *termination conditions* defined on special classes of multiset rewriting systems with linear constraints. We report here on practical experiments with coherence protocols for virtual shared memory, and multiprocessors systems in which the number of processors, pages or cache lines are left as parameters.

1 Introduction

Broadcast protocols [15] represent a very natural formalism to specify abstractions of *invalidation-based* protocols. In [16], Esparza, Finkel and Mayr have defined automated verification procedures to check *parameterized safety properties* (safety properties that are supposed to hold for any possible number of individual processes) for this class of infinite-state systems. The techniques of [16] exploit the property that the operational semantics of Broadcast Protocols can be given via extensions of Petri Nets with transfer arcs. *Backward reachability* always terminates for this class of Petri Nets when the seed of the computation consists of an upward-closed set of markings. The previous property reveals however a limitation of this model: processes must be represented here as indistinguishable *black tokens*. Thus, the algorithm of [16] can be applied only to analyze functional properties that can be formulated over *finite-state abstractions* of individual processes. As an example, in the models of cache coherence protocols given in [15, 16] each individual process (cache) is modeled via a finite-state automata, obtained by forgetting cache identifiers and by considering a single cache line and a single memory location. However, the original formulation of these protocols (see [5]) as well as of other invalidation-based protocols (see [20]) depends on several parameters like cache lines or entries in a page table. To obtain more *concrete* models, we need specification languages in which

individual processes are allowed to carry along information ranging over a possibly infinite domain, as in the formal models proposed in [18, 21]. In order to specify and validate these class of protocols, we will combine three different lines of research: *multiset rewriting* [11], *constraints* [14], and the *theory of well-quasi orderings* of [1, 17].

Multiset rewriting over first-order atomic formulas (MSR) [11] naturally extends the connection between rewriting and Petri Nets to nets with *colored tokens*, the colors being first-order terms attached to atomic formulas representing tokens. If we annotate rules with *constraints*, as proposed in [14], we achieve a clear separation between process structure and declaration of data relations, with advantages that will become clearer later. In this setting shared variables play the role of communication channels as shown in [10].

Suppose now we try to apply this paradigm to atomically specify invalidation phases as in the Broadcast Protocols model. We first note that the number of processes involved in a broadcast cannot be fixed a priori (it depends on the current configuration). However, by the *locality* of rewriting, a multiset rewriting rule allows us to specify only the behavior of a fixed number of processes.

As first contribution, we propose here an extension of the MSR specification language of [11] in which we introduce a mechanism for *global synchronization* that can be used to model broadcasts. To explain our idea, let us consider the *invalidation phase* caused by a *write miss* in the model of the M.E.S.I. cache coherence protocol (with a single cache line and memory location) given in [15]. In this example, every cache is modeled via a finite-state net. A token in a given place (inv, $modified$, $exclusive$, etc.) represents a cache in that state. On a write miss, the cache requesting to write its (unique) line first broadcasts a *signal* to invalidate all other caches (all other tokens move to place inv). In a Broadcast Protocol this behavior is captured by associating a set of *reactions* (e.g. cache invalidation) to a given action (e.g. broadcast sending). We obtain a similar effect in the MSR setting as follows. Let us model an action via a multiset rewriting rule, and any of the corresponding reactions as a rewriting rule $A \hookrightarrow B$ defined over two atomic formulas A and B. A reaction describes the behavior of a process in state A upon the reception of the broadcast. Using the syntax that we will formally introduce in Section 2, we formulate then the M.E.S.I. invalidation phase via rule like

$$inv \longrightarrow dirty\ [\ shared \hookrightarrow inv,\ dirty \hookrightarrow inv, \dots\ ]$$

(the set of reactions associated to an MSR rule is written between square brackets). Furthermore, we will extend the operational semantics so as to enforce the application of reactions to the maximal multiset of atomic formulas matching their left-hand side.

In the formalism we will propose in this paper, that we call $^{\hookrightarrow}\mathrm{MSR}(\mathscr{C})$, we will extend this idea to tokens colored with data. We achieve this aim by annotating an action and its reactions with a constraint (over a constraint system $\mathscr{C}$ that is a parameter of the language) defined over variables occurring in the corresponding atomic formulas. For instance, we refine the previous invalidation phase leaving

the number of cache lines as a parameter as follows

$$inv(p,m) \longrightarrow dirty(p,m)\ [\ shared(q,n)\ \hookrightarrow\ inv(q,n), \dots\ ] : p \neq q, m = n, \dots$$

where $p, q, m, n, \dots$ denote universally quantified variables (ranging over integers), indicating the identifiers of processors (p, q) and memory addresses (m, n).

Following [1], the method we use to check *parameterized properties* for the resulting models is based on *symbolic backward reachability*. In this paper we will focus on safety properties whose violations can be expressed via *upward-closed* sets of configurations. Our symbolic representation is based on the notion of *constrained configuration* introduced in [14], i.e., *multisets of first-order atomic formulas annotated with constraints*. Multisets represent minimal requirements over the distribution of tokens in the net, whereas constraints provide a natural symbolic representation for relations over data of different processes. Based on this *rich assertional language*, we provide *symbolic* operations needed to implement *backward reachability*, namely we define a *symbolic predecessor operator* for $\hookrightarrow$MSR($\mathscr{C}$)-specifications, which is sound and complete for any constraint system. Furthermore, we will show that it is possible to algorithmically verify *safety properties* for a class of $\hookrightarrow$MSR-specifications equipped with a subclass of linear integer constraints. Under the previous hypotheses and following the general theory of [1, 17], we obtain an extension of Broadcast Protocols with unbounded local data and value passing for which we can check algorithmically interesting safety properties.

Our work enlarges the target of the method presented in [10] as well as of its predecessors [16, 13] to new or simply more concrete examples of *invalidation-based* protocols. We report here on an interesting experiment with a coherence protocol for virtual shared memory proposed by Li and Hudak in [20] and previously analyzed in [18]. Using our technique, we have automatically verified the protocol in which the number of threads, processors, and pages of virtual memory are unbounded parameters. Finally, we have automatically validated coherence protocols for multiprocessors systems (M.S.I., M.E.S.I., Synapse) in which number of processors, cache lines and memory locations are left as parameters. In the following sections we will turn the previous intuitions into formal definitions.

2 Synchronous Multiset Rewriting

A *constraint system* is a tuple $\mathscr{C} = \langle \mathscr{V}, \mathscr{L}, \mathscr{D}, \mathscr{S} \rangle$ where: $\mathscr{V}$ is a denumerable set of variables; $\mathscr{L}$ is a *first-order* language with variables in $\mathscr{V}$ and closed with respect to *existential quantification*, *conjunction*, and with *equality*; $\varphi \in \mathscr{L}$ is called a *constraint*; $\mathscr{D}$ the *domain* of interpretation of constraints; a solution of a constraint φ is a mapping $\mathscr{V} \rightsquigarrow \mathscr{D}$ that satisfies φ; finally, $\mathscr{S}(\varphi)$ is the set of solutions of φ. A constraint φ is *satisfiable* whenever $\mathscr{S}(\varphi) \neq \emptyset$. Given a solution σ, we will use $\sigma_{|V}$ to indicate the restriction of σ to the variables in $V \subseteq \mathscr{V}$.

Let $\mathscr{P}$ be a set of symbols, an *atomic formula* $p(x_1, \dots, x_n)$ is such that $p \in \mathscr{P}$,

and $x_1, \ldots, x_n \in \mathscr{V}$ are *distinct* variables. A *ground* atomic formula $p(d_1, \ldots, d_n)$ is obtained by applying a solution σ to $p(x_1, \ldots, x_n)$ so that $\sigma(x_i) = d_i$ for $i : 1, \ldots, n$. A *multiset* of atomic formulas is indicated as $A_1 \mid \ldots \mid A_k$, where A_i and A_j have distinct variables, $\mid$ is an associative and commutative constructor, and ϵ denotes the empty multiset.

Definition 1. A *configuration* is a multiset of *ground atomic formulas*.

Intuitively, a configuration denotes a multiset of colored tokens. In the rest of the paper will use $\mathcal{M}$, $\mathcal{N}$, $\mathcal{H}$, $\mathcal{B}$, ... to denote *multisets* of atomic formulas.

Definition 2. An *action* is a rewrite rule $\mathcal{H} \longrightarrow \mathcal{B}$, where $\mathcal{H}$ and $\mathcal{B}$ are two multisets of atomic formulas with distinct variables. A *reaction* is a rewrite rule $A \hookrightarrow B$ where A and B are two atomic formulas with *distinct* variables.

Definition 3 (Rules). Given $\mathscr{C} = \langle \mathscr{V}, \mathscr{L}, \mathscr{D}, \mathscr{S} \rangle$, an $^{\hookrightarrow}\mathrm{MSR}(\mathscr{C})$ *rule* has the form $\mathcal{H} \longrightarrow \mathcal{B}\ [A_1 \hookrightarrow B_1, \ldots, A_k \hookrightarrow B_k] : \varphi$, where

1. $\mathcal{H} \longrightarrow \mathcal{B}$ is an action, and $A_i \hookrightarrow B_i$ is a reaction for $i : 1, \ldots, k$; action and reactions have distinct variables from each other;
2. the constraint $\varphi \in \mathscr{L}$ is such that $\varphi = \varphi_a \wedge \varphi_1 \wedge \ldots \wedge \varphi_k$, where φ_a is defined over the variables of the action, and the constraint $\varphi_i \in \mathscr{L}$ is defined over the variables of the action and of the i-th reaction $A_i \hookrightarrow B_i$ for $i : 1, \ldots, k$;
3. the constraint $A_i = B_j$ is not satisfiable for $i \neq j$;
4. the constraint $A_i = A_j \wedge \varphi_a \wedge \varphi_i \wedge \varphi_j$ is not satisfiable for $i \neq j$.

In the following $\mathcal{H} \longrightarrow \mathcal{B} : \varphi$ will denote a rule without reaction. The variables shared between an action and a reaction, and occurring in φ_i are the *communication channels* between the sender and a receiver. Condition 3 rules out cyclic reactions, and condition 4 ensures the determinism of reactions.

For instance, let be R the following *invalidation step* of a coherence protocol $invalid(p, m) \longrightarrow modified(p', m')\ [modified(q, n) \hookrightarrow invalid(q', n')] : \varphi$, where φ is defined as $p \neq q, m' = m = n' = n, p' = p, q' = q$; p, q can be interpreted as processor identifiers, and m, n as addresses of data stored in the local caches.

Definition 4. An $^{\hookrightarrow}\mathrm{MSR}(\mathscr{C})$ *specification* is a tuple $\langle \mathscr{P}, \mathscr{C}, \mathscr{I}, \mathscr{R} \rangle$, where $\mathscr{P}$ is a set of predicate symbols, $\mathscr{C}$ is a constraint system, $\mathscr{I}$ is a set of configurations (initial configurations), and $\mathscr{R}$ is a set of $^{\hookrightarrow}\mathrm{MSR}(\mathscr{C})$ rules built over $\mathscr{P}$.

Let $\oplus$ and $\ominus$ denote *multiset union* and *difference*, respectively. Furthermore, let $\preccurlyeq$ indicate the *multiset ordering*: $\mathcal{M} \preccurlyeq \mathcal{N}$ if $\zeta_A(\mathcal{M}) \leq \zeta_A(\mathcal{N})$ for every ground atom A, where $\zeta_A(\mathcal{M})$ is the number of occurrences of A in $\mathcal{M}$. The operational semantics of an $^{\hookrightarrow}\mathrm{MSR}(\mathscr{C})$ specification $\langle \mathscr{P}, \mathscr{C}, \mathscr{I}, \mathscr{R} \rangle$ is defined as follows.

Definition 5 (Enabling a Rule). A rule $\mathcal{H} \longrightarrow \mathcal{B}\ [A_1 \hookrightarrow B_1, \ldots, A_k \hookrightarrow B_k] : \varphi$ from $\mathscr{R}$ is *enabled at* a configuration $\mathcal{M}$ *via* $\sigma \in \mathscr{S}(\varphi)$ if $\sigma(\mathcal{H}) \preccurlyeq \mathcal{M}$.

Definition 6 (Firing a Rule). Suppose the rule R defined as $\equiv \mathcal{H} \longrightarrow \mathcal{B}\ [A_1 \hookrightarrow B_1, \ldots, A_k \hookrightarrow B_k] : \varphi$ is enabled at $\mathcal{M}$ via $\sigma \in \mathscr{S}(\varphi)$. Firing it at $\mathcal{M}$ yields the configuration $\mathcal{M}'$, written $\mathcal{M} \Rightarrow_R \mathcal{M}'$, if there exists $n \geq 0$ such that

1. $\mathcal{M} = \sigma(\mathcal{H}) \oplus (C_1 \mid \ldots \mid C_n) \oplus \mathcal{Q}$, and $\mathcal{M}' = \sigma(\mathcal{B}) \oplus (C'_1 \mid \ldots \mid C'_n) \oplus \mathcal{Q}$;
2. $C_1 \mid \ldots \mid C_n$ is the *maximal* multiset contained in $\mathcal{M} \ominus \sigma(\mathcal{H})$ such that: for every $i : 1, \ldots, n$ there exists $\sigma_i \in \mathscr{S}(\varphi)$ with $\sigma_{i|V} = \sigma_{|V}$, $V \subseteq \mathscr{V}$ being the set of variables of the *action* $\mathcal{H} \longrightarrow \mathcal{B}$, and there exists a *reaction* $A_{j_i} \hookrightarrow B_{j_i}$ $j_i \in \{1, \ldots k\}$, such that $\sigma_i(A_{j_i}) = C_i$ and $\sigma_i(B_{j_i}) = C'_i$.

The values associated to the variables occurring in the action are fixed by condition 1. Condition 2 allows the same reaction to be applied to different processes (atomic formulas) with possibly different values of the variables not in V, the only constraint being for the resulting mapping to satisfy the constraint φ. From condition 2, it follows that Q has no occurrences of atomic formulas that can be unified (so that all the resulting constraints are satisfied) with the left-hand side of a reaction, in other words every process ready to react *must react*.

Let $\mathcal{M}$ be $invalid(5, 2) \mid modified(1, 2) \mid modified(1, 3) \mid modified(3, 2)$. The rule R of the previous example is enabled at $\mathcal{M}$, and, since the reactions can be applied only for $m = n = 2$, when fired, it yields the new configuration defined as $modified(5, 2) \mid invalid(1, 2) \mid modified(1, 3) \mid invalid(3, 2)$.

We conclude this section defining the predecessor operator.

Definition 7 (Pre-Image). Let S be a set of configurations. The *predecessor* operator Pre is defined as $Pre(S) = \{\mathcal{M} \mid \mathcal{M} \Rightarrow_R \mathcal{M}',\ \mathcal{M}' \in S,\ R \in \mathscr{R}\}$.

3 A Consistency Protocol for Virtual Shared Memory

In this section we will focus our attention on the *broadcast distributed manager* protocol for maintaining a virtual shared memory consistent, proposed by Li and Hudak in [20]. In [18], Fisler and Girault modeled this protocol as a Colored Petri Net (CPN) parametric on numbers of threads, processors, and pages. They manually validated the parametric model, and they automatically validated some of its finite-state instances. Our aim will be to automatically validate a parametric model. In this protocol the virtual shared memory is organized into *pages*. Every processor has a *page table* used to maintain the status of each page relative to the processor. The status indicates the processor access rights to the page, namely *nil*, *read*, or *write* (that includes *read*), and whether or not the processor is the *owner* of the page. Intuitively, the owner is the last one that modified the page. Several processors may have read access to a page at once, whereas write access is exclusive. An important feature of this case-study is that the owner *changes over time*. The owner takes care of providing the page contents to any other processor that requests them and keeps track of all other processes that have read access to the page in a *copy-set*. Every processor must be told to invalidate its own copy before a write to the page occurs. Following [18], we consider a system in which: processors may contain several threads, but only one thread per processor is capable of faulting (this thread has it own handler); each processor has a single server; furthermore, broadcast is atomic (broadcast messages arrive before all other messages). When a processor wants to gain privileges it does not

Read and Write Faults

1. $\epsilon \longrightarrow rf(i,p) \ : \ true \quad$ 2. $\epsilon \longrightarrow wf(i,p) \ : \ true$

Read Handler

3. $rf(i,p) \mid pt(j,q,l,o,a) \longrightarrow hrp(i,p) \mid pt(j,q,l',o,a) \ :$
$i = j, p = q, l = \neg\mathbf{lck}, o = \neg\mathbf{own}, a = \mathbf{nil}, l' = \mathbf{lck}$

4. $hrp(i,p) \mid pt(k,q,l,o,a) \longrightarrow hre(i,p) \mid pt(k,q,l',o,a) \mid sr(k,p) \ :$
$k \neq i, p = q, l = \neg\mathbf{lck}, \ o = \mathbf{own}, l' = \mathbf{lck}$

5. $hre(i,p) \mid pt(j,q,l,o,a) \longrightarrow pt(j,q,p,l',o,a') \ :$
$i = j, p = q, l = \mathbf{lck}, \ l' = \neg\mathbf{lck}, \ a' = \mathbf{rd}$

Read Server

6. $sr(i,p) \mid pt(j,q,l,o,a) \longrightarrow pt(j,q,l',o,a') \ :$
$i = j, p = q, l = \mathbf{lck}, \ l' = \neg\mathbf{lck}, \ a' = \mathbf{rd}$

Write Miss

7. $wf(i,p) \mid pt(j,q,l,o,a) \longrightarrow hwp(i,p) \mid pt(j,q,l',o,a) \ :$
$i = j, p = q, l = \neg\mathbf{lck}, \ a = \mathbf{nil}, \ l' = \mathbf{lck}$

8. $wf(i,p) \mid pt(j,q,l,o,a) \longrightarrow hwp(i,p) \mid pt(j,q,l',o,a) \ :$
$i = j, p = q, l = \neg\mathbf{lck}, \ o = \neg\mathbf{own}, \ a = \mathbf{rd}, l' = \mathbf{lck}$

9. $wf(i,p) \mid pt(j,q,l,o,a) \longrightarrow hwi(i,p) \mid pt(j,q,l',o,a) \ :$
$i = j, p = q, l = \neg\mathbf{lck}, \ o = \mathbf{own}, \ a = \mathbf{rd}, l' = \mathbf{lck}$

10. $hwp(i,p) \mid pt(k,q,l,o,a) \longrightarrow hwi(i,p) \mid pt(k,q,l',o,a) \mid sw(k,q) \ :$
$k \neq i, p = q, l = \neg\mathbf{lck}, \ o = \mathbf{own}, l' = \mathbf{lck}$

Invalidation

11. $hwi(i,p) \mid pt(j,q,l,o,a) \longrightarrow pt(j,q,l',o',a') \ [pt(k,r,m,s,b) \hookrightarrow pt(k,q,m,s,b')] :$
$i = j, p = q, l = \mathbf{lck}, l' = \neg\mathbf{lck}, \ o' = \mathbf{own}, \ a' = \mathbf{wrt}, k \neq i, \ p = r, \ b' = \mathbf{nil}$

Write Server

12. $sw(i,p) \mid pt(j,q,l,o,a) \longrightarrow pt(j,q,l',o',a') \ :$
$i = j, \ p = q, \ l = \mathbf{lck}, \ o = \mathbf{own}, l' = \neg\mathbf{lck}, \ o' = \neg\mathbf{own}, \ a' = \mathbf{nil}$

Fig. 1. Broadcast Distributed Manager

have, it faults and invokes a *handler* to request the page. Requests are broadcast to all other processors. The owner responds running a *server* while entries in the page table are locked to avoid conflicts between handlers and servers on the same processor.

In Fig 1 we illustrate an $\hookrightarrow$MSR-specification based on *linear arithmetic constraints* where conjunctions are like $x > z, x > y, t = 3$. The logic of the specification follows the CPN model of [18] from which we borrow the names of some places as described below. *Faulting threads* (the only one being interesting) are represented via atomic formulas $rf(i,p)$ (a read fault on processor i relative to page p) and $wf(i,p)$ (a write fault). Furthermore, entries in the page table of processor i will be represented via the atomic formula $pt(i,p,l,o,a)$, p being the page identifier, $l \in \{\mathbf{lck}, \neg\mathbf{lck}\}$ denoting whether the page entry for p is locked on i; $o \in \{\mathbf{own}, \neg\mathbf{own}\}$ denoting whether or not processor i is the owner of

p; and $a \in \{\mathbf{nil}, \mathbf{rd}, \mathbf{wrt}\}$ denoting the access rights of i for page p. Page contents are not relevant for the properties we consider here (mutual-exclusion). All free variables range over integers and may get unbounded values in a derivation starting from *init*, whereas the previous constants (**lck** etc.) must be read as fixed integer values. In the CPN model of [18] each entry of a page table has a local copy-set. However, as formally proved on the CPN model in [18], only the copy-set of the owner is of interest for the logic of the protocol. Taking this assumption, we model the *copy-set* via the multiset of entries with **rd** access rights contained in the current configurations. Only the current owner can modify it.

The first block of rules in Fig 1 non-deterministically simulates *write* and *read* faults. The second and third block define the behavior of read handlers and servers. On a read fault rf, the processor invokes the *read handler* (hrp=handler-read-prepare) that takes care of contacting the current owner of the page. Before terminating its execution, the (hre=handler-read-end) the handler modifies the rights of the processor entry page. The *read server* (modeled as sr) takes care of updating the rights of the owner entry page. The last blocks define the behavior of write handlers and servers. On a write fault wf, we distinguish two cases. If the processor on which the processor runs is the owner, then the thread directly goes to the invalidation state hwi (this avoids Error 2 of [18]). Otherwise, the processor invokes a *write handler* (hwp=handler-write-prepare) and locks the entry of the page table, while the handler moves to the invalidation-phase (hwi=handler-write-invalidate). At this stage the handler acquires the ownership of the page, updates his rights to *write*, while invalidating all processors reading that page. We model the invalidation-phase atomically using an action-reaction rule. The *write server* (modeled as sw) of the owner downgrades its rights to the page, relinquishes the ownership (this avoids Error 1 of [18]) of the page and then terminates. The initial states $\mathscr{I}$ are the configurations in which one processor is the owner (with read access) of all pages. Note that in our specification the values of the processor identifiers and pages are left unbounded.

The *safety property* we would like to establish on our specification is *mutual exclusion*. Specifically, we would like to ensure that the *write* access is *exclusive* for *any number of threads, processors, and pages*. We will come back to this verification problem after describing our method to handle this kind of parameterized invalidation-based protocols with unbounded local data.

4 Symbolic Verification Procedures

A parameterized safety property like mutual exclusion for our case study holds if and only if $\mathscr{I} \cap Pre^*(U) = \emptyset$, where U is the infinite collection of *unsafe* configurations (infinite because of the number of processes and values of local data). In our example (as in many other practical situations [1, 16, 13]) unsafe states turn out to be *upward-closed* w.r.t. multiset inclusion $\preccurlyeq$.

Definition 8. A set of configurations S is *upward-closed* whenever $Up(S) = S$, where $Up(S) = \{\mathcal{N} \mid \mathcal{M} \preccurlyeq \mathcal{N},\ \mathcal{M} \in S\}$.

To finitely represent the *generators* of an upward closed set of configurations, in [14] we introduced the notion of *constrained configuration* whose *rich denotation* consists of the *upward closure* of its *ground instances*. As an example, the *violations* to readers-writers mutual exclusion for Li-Hudak's protocol can be represented via

$$pt(i,p,l,o,a) \mid pt(k,q,m,s,b) \ : \ i \neq k, p = q, a = \mathbf{rd}, b = \mathbf{wrt}$$

whose denotations are: $(pt(u,v,w,y,\mathbf{rd}) \mid pt(u',v',w',y',\mathbf{wrt})) \ \oplus \ \mathcal{M}$ for any integers $u, v, u', v', \ldots \in \mathbb{Z}$, with $u \neq u'$, and for any other configuration $\mathcal{M}$. This example explains the reason why we introduced constraints in our specification language: they provide a natural symbolic representation for relations over data of different processes. Constrained configurations are defined as follows.

Definition 9 (Constrained Configuration). A *constrained configuration* has the form $p_1(x_{11},\ldots,x_{1k_1}) \mid \ldots \mid p_n(x_{n1},\ldots,x_{nk_n}) : \varphi$ where $p_1,\ldots,p_n \in \mathscr{P}$, $x_{i1},\ldots,x_{ik_i} \in \mathscr{V}$ for $i : 1,\ldots n$ and $\varphi \in \mathscr{L}$; the variables $x_{11},\ldots$ are distinct each other.

The *set* of *ground instances* of a constrained configuration $\mathcal{M} : \varphi$ is defined as $Inst(\mathcal{M} : \varphi) = \{\sigma(\mathcal{M}) \mid \sigma \in \mathscr{S}(\varphi)\}$. The previous definition can be extended to *sets of constrained configurations* with *disjoint variables*, written $\mathbf{S}$, $\mathbf{S}'$, etc., in the natural way. *Rich denotations* are built as follows.

Definition 10 (Rich Denotation). Given a set $\mathbf{S}$ of constrained configurations (with disjoint variables), its denotation is defined as $[\![\mathbf{S}]\!] = Up(Inst(\mathbf{S}))$.

We will define next a symbolic pre-image operator **Pre** working on our assertional language according to the *rich* denotation of constrained configurations. Let us first introduce the notion of *unification* between two multisets of atomic formulas (with disjoint variables). The relation $(A_1 \mid \ldots \mid A_n) =_\theta (B_1 \mid \ldots \mid B_m)$ holds provided $m = n$ and the constraint $\theta = \bigwedge_{i=1}^{n} A_i = B_{j_i}$ is *satisfiable*, $j_1,\ldots,j_n$ being a permutation of $1,\ldots,n$. Finally, a *variant* of an formula is obtained by renaming its free variables with *fresh names*. We are ready now to define the operator **Pre**.

Definition 11 (Symbolic Pre-image). Let $\mathbf{S}$ be a set of $\mathscr{C}$-constrained configurations. The symbolic predecessor operator **Pre** is defined as follows: $(\mathcal{N} : \xi) \in \mathbf{Pre}(\mathbf{S})$ if and only if there exists a *variant* $(\mathcal{M} : \psi)$ of a constrained configuration in $\mathbf{S}$, a *variant* $\mathcal{H} \longrightarrow \mathcal{B}\ [\ A_1 \hookrightarrow B_1,\ldots,A_k \hookrightarrow B_k\ ] : \varphi$ of a rule in $\mathscr{R}$ (recall that $\varphi = \varphi_a \wedge \varphi_1 \wedge \ldots \wedge \varphi_k$ as from Def. 3), and multisets $\mathcal{M}', \mathcal{B}', \mathcal{Q}$ and $n \geq 0$ such that

1. $\mathcal{M} = \mathcal{M}' \oplus (C_1 \mid \ldots \mid C_n) \oplus \mathcal{Q}$;
2. $\mathcal{B}' \preccurlyeq \mathcal{B}$ and $\mathcal{M}' =_\theta \mathcal{B}'$;
3. for all $i : 1,\ldots,n$ there exist $j_i \in \{1,\ldots,k\}$ and *variants* $A'_{j_i} \hookrightarrow B'_{j_i}$ and φ'_{j_i} of a reaction $A_{j_i} \hookrightarrow B_{j_i}$ and of the associated constraint φ_{j_i} (both obtained via the renaming ι_i) such that: $C_{j_i} =_{\theta_i} B'_{j_i}$ holds, the constraint $\gamma_i \equiv \psi \wedge \theta \wedge \varphi_a \wedge \varphi'_{j_i} \wedge \theta_i$ is satisfiable;

4. Q does not contain occurrences of atomic formulas D for which there exist $q \in \{1, \ldots, k\}$ (i.e. a reaction $A_q \hookrightarrow B_q$) such that $D =_\tau A_q$ holds, and $\psi \wedge \theta \wedge \varphi_a \wedge \varphi_q \wedge \tau$ is satisfiable;
5. $\mathcal{N} = \mathcal{H} \oplus (A'_{j_1} \mid \ldots \mid A'_{j_n}) \oplus \mathcal{Q}$;
6. the constraint ξ defined as $\exists x_1 \ldots . \exists x_u . \gamma_1 \wedge \ldots \wedge \gamma_n$ is satisfiable, where $x_1, \ldots, x_u$ are all the variables of $\gamma_1, \ldots, \gamma_n$ that do not occur in $\mathcal{N}$.

The symbolic operator **Pre** returns a set of constrained configurations and it is correct and complete with respect to Pre, i.e., $[\![\mathbf{Pre}(\mathbf{S})]\!] = Pre([\![\mathbf{S}]\!])$ for any $\mathbf{S}$.

As an example, consider the constrained multiset $M \equiv (\mathcal{M} : \varphi)$, where $\mathcal{M}$ is the multiset $modified(p, m) \mid modified(q, n)$, and φ is the constraint $m = n$ and the rule R we introduced in previous examples. Then, by selecting $\mathcal{M}' = \epsilon$ and $\mathcal{B}' = \epsilon$ in Def. 11, we obtain M itself. However, all other attempts of matching R with M fail (M does not satisfy the maximality of reaction applications). Contrary, $(invalid(r, l) \mid invalid(p, m) \mid modified(q, n) : l = m = n)$ has several predecessors like $(modified(r, l) \mid modified(p, m) \mid invalid(q, n) : l = m, m = n)$ as well as $(invalid(r, l) \mid modified(p, m) \mid invalid(q, n) : l = m = n)$; the latter expresses the fact that cache r was already invalid before firing R.

To define a symbolic reachability algorithm, we still need a comparison operator between constrained configurations.

Definition 12. An *entailment* $\sqsubseteq$ between constrained configurations is a relation such that $M \sqsubseteq N$ implies $[\![N]\!] \subseteq [\![M]\!]$.

We can now rephrase *backward reachability* as follows. Let $\mathbf{U}$ be a set of constrained configurations. We first compute $Pre^*(\mathbf{U})$: starting from $\mathbf{U}$, we repeatedly apply **Pre** to all stored constrained configurations. We stop when it is not possible to store new constrained configurations (i.e. for each new constrained configuration M we already computed N such that $N \sqsubseteq M$). If the fixpoint computation terminates we check that the initial configurations representing the *initial states* of the system are not contained in the denotation of the resulting set of constrained configurations (e.g. $init \notin [\![\mathbf{Pre}^*(\mathbf{U})]\!]$).

5 Sufficient Conditions for Termination

As shown in Section 3, our examples make use of arithmetic constraints. Thus, the termination of the symbolic backward reachability procedure cannot be guaranteed in general. To obtain sufficient conditions for termination, we need severe restrictions on the form of constraints and rewrite rules we allow in the specification. Let us first introduce the class NC of linear constraints defined as conjunctions of atomic formulas either of the form $x > y$ or $x = y$. Let us assume that all predicates in $\mathscr{P}$ have arity n. Given a constrained configuration $\mathcal{M} : \varphi$ let V_i be the set of variables occurring in position i in atoms of M for $i : 1, \ldots, n$. Finally, let $Var(\varphi)$ be the set of variables occurring in φ. Then, we further restrict NC-constrained configurations as follows.

Definition 13. An NC-constrained configuration $(\mathcal{M} : \varphi)$ is k-separable if φ can be partitioned into $\varphi_1, \ldots, \varphi_n$ such that $Var(\varphi_i) \subseteq V_i$ for $i : 1, \ldots, n$, and $x = y$ follows from φ_i for all $x, y \in V_i$, $i \neq k$.

Without loss of generality, we restrict our attention to k-separable constrained configurations in which all variables in position k (thus, the corresponding atomic formulas) are *totally ordered* w.r.t. to the $=$ and $>$ relations induced by φ_k. Given a k-separable constrained configuration C defined as $(\mathcal{M} : \varphi)$, C can be uniquely represented as a *string* $Str(C)$ of *multisets* of symbols in $\mathscr{P}$ built as follows. We first group together all atomic formulas in which the corresponding variables in position k are related via equality constraints. Then, we order the resulting groups according to the $>$ relation induced by φ_k. Since φ is satisfiable, we obtain an acyclic path. Finally, we represent every group of atomic formulas as the multiset obtained by selecting the corresponding predicate symbols.

For instance, given $C = p(a, x) \mid p(b, y) \mid q(c, z) : a = b = c, x > y, y = z$ we first group together the atomic formulas as follows: $C_1 = \{p(a, x)\}$, and $C_2 = \{p(b, z), q(c, y)\}$. $S(C)$ is then the string of multisets: $p \cdot pq$.

Definition 14. Given two k-separable constrained configurations C and D such that $S(C) = \mathcal{M}_1 \cdot \ldots \cdot \mathcal{M}_k$ and $S(D) = \mathcal{N}_1 \cdot \ldots \cdot N_r$, let $C \sqsubseteq^* D$ iff there exists an *injective* mapping h from $1, \ldots, k$ to $1, \ldots, r$ such that if $i < j$ then $h(i) < h(j)$, and $\mathcal{M}_i \preccurlyeq \mathcal{N}_{h(i)}$ for $i : 1, \ldots, k$ ($\preccurlyeq$ denotes *multiset inclusion*).

It turns out that $\sqsubseteq^*$ is an entailment. Furthermore, following from the properties of the multiset and string embeddings [19], $\sqsubseteq^*$ is a *well-quasi-ordering*. Now, let us impose the following restriction on $\hookrightarrow$MSR(NC)-specifications.

Definition 15. A *k-separable specification* is such that the multisets occurring in the action and reactions of a rule $\mathcal{H} \longrightarrow \mathcal{B}\ [A_1 \hookrightarrow B_1, \ldots, A_n \hookrightarrow B_n] : \varphi$, are k-separable constrained configurations w.r.t. $\varphi_a, \varphi_1, \ldots, \varphi_n$.

Then, we obtain the following results.

Theorem 1. The class of k-separable constrained configuration is closed under applications of **Pre** associated to a k-separable specification. The symbolic backward reachability algorithm always terminates when taking in input a k-separable specification and a set of k-separable constrained configurations.

6 Experimental Results

We have implemented the backward reachability algorithm using the SICStus term manipulation and constraint solving libraries (e.g. for linear constraints). We used our prototype to analyze the Li-Hudak protocol of Fig. 1. We recall that our specification is a reformulation of the CPN model of [18] formally validated via static analysis and model checking. However, as shown in Fig. 2, we automatically discovered after 11 steps the following *error-trace*:

1. on a read fault a thread t (not running on the owner) invokes its handler and then suspends;

Model	Verification Problem	Steps	Size	Time	Verified?
Li-Hudak model Fig. 1	Read/write mutex	11	3600	6609s	Error found
Correct Li-Hudak model	Read/write mutex	7	24	0.8s	Yes
M.S.I. with many cache lines	Read/write mutex	1	2	0.01s	Yes
Synapse N+1 "	Read/write mutex	1	2	0.01s	Yes
M.E.S.I. "	Read/write mutex	3	7	0.1s	Yes

Fig. 2. Experimental results on a Pentium II 450Mhz.

2. the server of the owner grants the request and updates its entry page;
3. a thread s running on the owner faults for write access;
4. the owner grants it;
5. the handler of s invalidates all other processors and the owner's rights upgrade to *write*;
6. finally, the suspended thread t updates its entry page to *read*.

This error trace uncovers a violation to writer exclusivity in the CPN model of [18]. From personal communication, Kathi Fisler informed us that this model was validated only manually (via static analysis). The analysis uncovered other errors but not the previous critical one. The models automatically verified with Cospan and Murϕ in [18] were based instead on different assumptions on inter-processor communication. Interestingly, the error we uncovered matches the Error 3 of [18] that Fisler and Girault only discovered using a more refined CPN model with explicit queue channels. However, the previous error occurs even in the more abstract model: it is caused, in fact, by a missing synchronization between handlers and servers. The nature of this error also reveals an ambiguity in the original, informal specification [20]: the notation used by Li and Hudak does not clarify, in fact, whether or not a broadcast should be *blocking* for the sender. This case study reveals once again the difficulties in using manual abstractions (i.e. to pass from the CPN model to another one) when modeling complex protocols.

To correct the error, the handler must wait for an acknowledgment from the server. We have encoded the corrected model in our framework, and we have automatically verified mutual exclusion (we computed the fixpoint in 7 steps and 0.8s on a Pentium II 450Mhz) as well as other properties (e.g. the owner (of unlocked pages) is unique, there is only one page entry for each page and node), whose violations can be naturally expressed in our assertional language) for an *arbitrary number* of *threads*, *processors*, and *pages*.

Finally, we have also modeled and automatically verified mutual exclusion for three cache coherence protocols for multi-processor systems (M.S.I., Synapse, and M.E.S.I. see Fig. 2) that, differently from previous models given via Broadcast protocols [15, 16, 13], we formulated for an *arbitrary number* of cache and memory lines (see the rule example of the introduction).

7 Conclusions and Related Works

With this work we have enlarged the scope of parameterized verification to invalidation-based protocols formulated at a greater level of detail than Broadcast Protocols: processes may have here unbounded local data and communicate via value passing. This way, we were able to analyze and observe interesting properties in different formal models of cache coherence protocols.

The model presented here can neither be formulated nor analyzed in the framework we proposed in [10], in which only rendez-vous is allowed as synchronization mechanism for colored tokens. Furthermore, contrary to what we claimed in [10], the class of separable specifications isolated in [10] must be futher restricted (e.g. using the notion of k-separability) to obtain the termination of backward reachability. Together with the automated analysis of a parameterized formulation of the *ticket algorithm* considered in [10], the preliminary results of our work show the potential interest of this line of research. Another possible way to model invalidation-phases in multiset rewriting would be to introduce tests for the emptiness of a place as in Gamma [21] or as in Petri Nets with inhibitor arcs. However, in [21] the authors need to use conservative *counting* abstractions to validate their Gamma specifications. In [21] they apply this method to verify a manually constructed abstract model of Li-Hudak protocol, in which, e.g., they do not distinguish between owners and writers. At this level of abstraction it is not possible to uncover the ambiguities in the original informal specification. In [8] a combination of manual abstractions, theorem proving (PVS), and automated abstractions (PAX [7]) has been used to verify Li-Hudak's protocol. The authors considered assumptions different from those taken in the first CPN model of [18]. Our work is inspired to the approach of [2, 4], where *existential regions* are proposed as symbolic representation of configurations for parameterized Timed Petri Nets. In [3], Abdulla and Jonsson have used similar techniques to prove termination for backward reachability of *Unordered Channel Systems* in which messages can vary over an infinite *name* domain. However, they do not provide any mechanism for invalidation-phases. Networks of *finite-state* processes can be analyzed using the automata theoretic approach of [9, 12, 23], where sets of global states are represented as *regular languages*, and transitions as relations on languages. Symbolic exploration can then be performed using operations over automata with ad hoc accelerations (see e.g. [9]), or with automated abstractions techniques (see e.g. [7]).

Our work is complementary to the approach based on the *deductive method* with *invisible invariants* of [6], in which invariants are first generated and then proved to be inductive. We follow here, in fact, the paradigm of *symbolic model checking with rich assertional languages* [22]. The two approaches can be used to attack similar problems using different point-of-views.

As future work we plan to investigate techniques for handling liveness properties, more realistic broadcast with message queues, and more complex properties like sequential consistency.

Acknowledgements We would like to thank Moshe Vardi for having helped us to find new directions in our research, and Kathi Fisler for fruitful discussions.

References

1. P. A. Abdulla, K. Cerāns, B. Jonsson, Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. LICS'96*, pp. 313–321, 1996.
2. P. A. Abdulla, B. Jonsson. Verifying Networks of Timed Processes (Extended Abstract). In *Proc. TACAS '98*, pp. 298–312. Springer, 1998.
3. P. A. Abdulla, B. Jonsson. Channel Representations in Protocol Verification. In *Proc. CONCUR 2001*, pp. 1–15. Springer, 2001.
4. P. A. Abdulla, A. Nylén. Better is Better than Well: On Efficient Verification of Infinite-State Systems. In *Proc. LICS 2000*, pp. 132–140, 2000.
5. P. A. Archibald, J. Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. *TOCS* 4(4): 273–298. 1986.
6. T. Arons, A. Pnueli, S. Ruah, Y. Xu, L. Zuck Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. CAV '01*, pp. 221-234, 2001.
7. K. Baukus, S. Bensalem, Y. Lakhnech, K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In *Proc. TACAS '00*, pp. 188-203, 2000.
8. K. Baukus, K. Stahl, S. Bensalem, Y. Lakhnech. Networks of Processes with Parameterized State Space. In *Proc. VEPAS '01*, ENTCS vol. 50, issue 4, 2001.
9. A. Bouajjani, B. Jonsson, M. Nilsson, T. Touilli. Regular Model Checking. In *Proc. CAV'00*, pp. 403-418, 2000.
10. M. Bozzano, G. Delzanno. Beyond Parameterized Verification. In *Proc. TACAS '02*, April 2002.
11. I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, A. Scedrov. A Meta-notation for Protocol Analysis. In *Proc. CSFW 1999*, pp. 55–69, 1999.
12. E. Clarke, O. Grumberg, S. Jha. Verifying Parameterized Networks. *TOPLAS* 19(5): 726–750, 1997.
13. G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In *Proc. CAV 2000*, pp. 53–68, 2000.
14. G. Delzanno. An assertional language for systems parametric in several dimensions. In *Proc. VEPAS 2001*, ENTCS 50(4), 2001.
15. E. A. Emerson, K. S. Namjoshi. On Model Checking for Non-deterministic Infinite-state Systems. In *Proc. LICS '98*, pp. 70–80, 1998.
16. J. Esparza, A. Finkel, R. Mayr. On the Verification of Broadcast Protocols. In *Proc. LICS'99*, pp. 352–359, 1999.
17. A. Finkel, P. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):63–92, 2001.
18. K. Fisler, C. Girault. Modelling and Model Checking a Distributed Shared Memory Consistency Protocol. In *Proc. ICATPN '98*, pp. 84-103, 2001.
19. G. Highman. Ordering by Divisibility in Abstract Algebras. *Proc. London Math. Soc.*, 2:326–336, 1952.
20. K. Li, P. Hudak. Memory Coherence in Shared Virtual Memory Systems. TOCS 7(4): 321-359, 1989.
21. D. Mentré, D. Le Métayer, T. Priol. Formalization and Verification of Coherence Protocols with the Gamma Framework. In *Proc. PDSE 2000*, pp. 105-113, 2000.
22. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *Proc. CAV'97*, pp. 424–435, 1997.
23. A. P. Sistla, V. Gyuris. Parametrized Verification of Linear Networks using Automata as Invariants. *Formal Aspects of Computing*, 11(4):402–425, 1999.