Automatic Verification of Parameterized Cache Coherence Protocols

Giorgio Delzanno

DISI - University of Genova
via Dodecaneso 35, 16146 Italy
delzanno@disi.unige.it

Abstract. We propose a new method for the verification of *parameterized* cache coherence protocols. Cache coherence protocols are used to maintain data consistency in multiprocessor systems equipped with local fast caches. In our approach we use arithmetic constraints to model possibly infinite sets of global states of a multiprocessor system with many identical caches. In preliminary experiments using symbolic model checkers for *infinite-state* systems based on real arithmetics (HyTech [HHW97] and DMC [DP99]) we have automatically verified *safety* properties for parameterized versions of widely implemented *write-invalidate* and *write-update* cache coherence policies like the Mesi, Berkeley, Illinois, Firefly and Dragon protocols [Han93]. With this application, we show that symbolic model checking tools originally designed for hybrid and concurrent systems can be applied successfully to a new class of *infinite-state* systems of practical interest.

1 Introduction

In a shared-memory multiprocessor system *local caches* are used to reduce memory access latency and network traffic. Each processor is connected to a fast memory backed up by a large (and slower) main memory. This configuration enables processors to work on local copies of main memory blocks, greatly reducing the number of memory accesses that the processor must perform during program execution. Although local caches improve system performance, they introduce the *cache coherence problem*: multiple cached copies of the same block of memory must be consistent at any time during a run of the system. A *cache coherence protocol* ensures the *data consistency* of the system: *the value returned by a read must be always the last value written to that location* (cf. [AB86, Han93, PD95]). Coherence policies can be described as *finite state machines* that specify the way a single cache reacts to read and write requests. As an example, let us consider a CC-UMA (Uniform-Memory-Access with local Caches model) multiprocessor system, i.e., a system in which all processors have a local cache connected to the main memory via a shared bus. In *write-invalidate* protocols, whenever a processor modifies its cache block a *bus invalidation signal* is sent to all other caches in order to invalidate their content. Instead, in *write-update* protocols a copy of the new data is sent to all caches that share the old data.

Due to the increasing complexity of hardware architectures, the development of *automatic* verification techniques is becoming a major goal to help discovering errors at an early stage of protocol design (see e.g. [CGH$^+$93, MS91]). In particular, one of the main challenges in this area is to develop techniques for validating protocols for *every possible number* of processors (see e.g. [EN98, HQR99, PD95]). In this paper, drawing inspiration from recent works on verification of *parameterized* concurrent systems (e.g. [GS92, EN96, EN98, EN98b, EFM99, LHR97]) we propose a new method for the verification of *parameterized* cache coherence protocols at the *behavior* (specification) level. As mentioned before, in this context a multiprocessor system can be modeled as a collection of many *identical* finite-state machines. As first step, we apply the following *abstraction*: we keep track only of the *number* of caches in every possible protocol state. The resulting abstract protocol can be represented as a transition system with data variables ranging over positive integers. Thus, an abstract protocol can be formally described as an Extended Finite State Machine (EFSM) [CK97]. Via this abstraction, we represent all *symmetric* global states (global state=collection of individual cache states) using a single EFSM-state. We use then arithmetic constraints to implicitly represent (potentially infinite) sets of EFSM-states (tuples of natural numbers). This way, we are able to represent *safety* properties independently from the number of processors, and we reduce the verification problem for parameterized cache coherence protocols to a *reachability problem* for EFSMs. The last problem can be attacked using *general purpose*, *infinite-state* symbolic model checking methods defined for integers or real arithmetics (see e.g. [BGP97, BW98, Hal93, HHW97, DP99]). Following the general methodology we suggest in [Del00, DP99], we apply efficient tools based on *real* arithmetics (thus, applying a *relaxation* from integers to reals during the analysis) to *automatically* check safety properties like *data-consistency* for *snoopy*, *write-invalidate* and *write-update* cache coherence protocols for CC-UMA multiprocessors [AB86, Han93, PD97].

More precisely, our contributions are as follows. We first show that parameterized versions of a large class of cache coherence protocols can be formulated in terms of EFSMs. The class of EFSMs we consider is an extension of the broadcast protocols of Emerson and Namjoshi [EN98]. In order to model coherence policies, e.g., like the Illinois protocol, without abstracting away properties that are crucial for their validation (see discussion in Section 5), we need to enforce *global conditions* that cannot be represented using broadcast protocols. To prove the adequacy of this encoding, we relate the EFSM model to the *finite state machine model* of cache coherence protocols proposed by Pong and Dubois [PD95]. Based on this idea, we define a general method for the validation of parameterized coherence protocols. The method is based on invariant checking for the corresponding EFSMs via the *backward reachability* algorithm of [ACJT96]. In contrast to *forward* reachability, the algorithm of [ACJT96] is guaranteed to terminate for the subclass of EFSMs denoting the broadcast protocols of [EN98] under the additional hypothesis that the set of unsafe states is upward-closed [EFM99]. Safety properties can often be modeled as upward-closed sets

[AJ99]. We choose a symbolic representation of (potentially infinite) sets of states via *arithmetic constraints*. The constraint operations of *variable elimination*, *satisfiability* and *entailment test* can be used to implement a symbolic version for the algorithm of [ACJT96]. However, following [DP99], in order to obtain an *efficient* procedure we interpret the above mentioned constraint operations over *reals* instead that over *integers*. This *relaxation* technique is widely-used in integer programming and program analysis. As for other methods handling *global conditions* in parameterized systems (e.g. [ABJN99]) and other methods for infinite-state systems (e.g. [BGP97, BW98, DP99, HHW97]), the resulting procedure is a *semi-algorithm* that must be evaluated on practical examples. We give sufficient conditions for the termination of the resulting procedure. Specifically, we show that the symbolic version of the abstract algorithm of [ACJT96] where sets of states are represented as arithmetic constraints is *robust under the relaxation integer-reals* (it always terminates solving the *control reachability problem* of [AJ99]) whenever: (a) the input EFSM is a broadcast protocol [EN98]; (b) the unsafe states are represented via a special class of constraints that denote upward-closed sets. This result seems to be a new application of general methods for proving the well-structuredness of infinite-state systems [ACJT96, AJ99, FS98]. We use two existing constraint-based model checkers that implement the symbolic backward reachability algorithm described above, namely HyTech [HHW97] (that provides efficient data structures) and DMC [DP99] (that provides built-in accelerations), to check several safety properties for the MESI, University of Illinois, Berkeley RISC, DEC Firefly and Xerox PARC Dragon protocols [AB86, PD95, Han93]. Though the termination of our method is guaranteed only for broadcast protocols, the preliminary results show that it performs well in practice.

To our knowledge, this is the first time that *general purpose* symbolic model checkers for infinite-state systems working over arithmetical domains are used for verification of parameterized cache coherence protocols. With this application, we have shown that techniques developed in the last years for hybrid and concurrent systems can also be applied to a new class of infinite-state systems of practical interest.

The HyTech and DMC code of the protocols together with the results of their analysis and links to download the tools is available on the web at the following address: `http://www.disi.unige.it/person/DelzannoG/protocol.html`.

2 The Finite State Machine Model

According to [PD95, PD97, EN98], we limit ourselves to consider protocols controlling single memory blocks and single cache lines. Following [PD95], a cache coherence protocol for a multiprocessor system with k local caches $C_1, \ldots, C_k$ can be represented via the following *finite state* machine model.

Local Machine. Each of the caches has the same finite set Q of states. The transitions of cache C_i may be guarded by *global conditions* that depend on the state of the other caches. The global conditions from the perspective of C_i are represented via a predicate f_i. As an example of global condition, let us fix a state $q \in Q$. Then, we could let $f_i = true$ if only if in the current state of the system there exists a cache C_j $(j \neq i)$ whose state is equal to q. Formally, the behavior of the cache C_i is represented as a finite system $\langle Q, \Sigma_i, \delta_i, f_i \rangle$, where Q is the set of states, Σ_i is the set of operations causing state transitions, $f_i : Q^k \to \{true, false\}$ is a predicate that represents the global conditions from the perspective of C_i, and δ_i defines the state transition $Q \times \Sigma_i \times \{true, false\} \to Q$. The third component in the domain of δ_i is the guard for the transitions of C_i. As an example, let us fix a state $q \in Q$, and an operation $\sigma \in \Sigma_i$. Then, we could set $\delta_i(q, \sigma, true) = q'$ to express that cache C_i can go from state q to state q' whenever f_i is satisfied in the current global state. The previous definitions allow us to compose the machines of the individual caches $C_1, \ldots, C_k$ into a single global machine $\mathcal{M}_G$.

Global Machine. A global state G of $\mathcal{M}_G$ is defined as the composition of the states of the individual caches. Formally, $\mathcal{M}_G$ is a tuple $\langle Q_G, \Sigma_G, \mathcal{F}, \delta_G \rangle$, where $Q_G = Q^k$, $\Sigma_G = \Sigma_1 \cup \ldots \cup \Sigma_k$, $\mathcal{F}$ is the global characteristic predicate $\langle f_1, \ldots, f_k \rangle$, and $\delta_G : Q_G \times \Sigma_G \to Q_G$. The transition function δ_G is defined as follows. Given a global state $G = \langle q_1, \ldots, q_k \rangle$, $\delta_G(G, \sigma) = \langle q'_1, \ldots, q'_k \rangle$ if and only if $q'_i = \delta_i(q_i, \sigma, f_i(G))$ for $i : 1, \ldots, k$. A run of $\mathcal{M}_G$ is a possibly infinite sequence of global states $G_1, \ldots, G_n \ldots$ where $\delta_G(G_n, \sigma) = G_{n+1}$ for some $\sigma \in \Sigma$. We write $G \xrightarrow{*} G'$ to denote the existence of a run that goes from G to G'.

Terminology. In the rest of the paper we use the state *invalid* to denote either that the cache has no data or that its content has been invalidated. Cache coherence protocols implement the following basic operations from the perspective of cache C_i ($state(C_i) \in Q$ denotes its current state): *Read Miss*, a read request is sent to C_i and $state(C_i) = invalid$; *Read Hit*, a read request is sent to cache C_i and $state(C_i) \neq invalid$; *Write Miss*, a write request is sent to C_i and $state(C_i) = invalid$; *Write Hit*, a write request is sent to C_i and $state(C_i) \neq invalid$. According to the previous definitions, in the next section we give a brief description of a widely implemented *snoopy*, *write-invalidate* protocol we selected as main case-study.

2.1 The Illinois Protocol

The University of Illinois protocol is a *snoopy* cache, *write-invalidate*, *write-in* coherence policy, originally proposed by Papamarcos and Patel [PP84] (see also [AB86, PD95]). The special feature is that caches can have exclusive copies of data. Bus invalidation signals are sent only for writes to shared data. The memory copy is updated using a write-back policy (*replace* operation). Formally, in addition to *invalid*, caches assume one of the following states: *valid-exclusive*, the cache has an exclusive copy of the data that is consistent with the memory such that a modification of its content requires no bus invalidation signal; *shared*, the cache has a copy of the data consistent with the memory and other caches

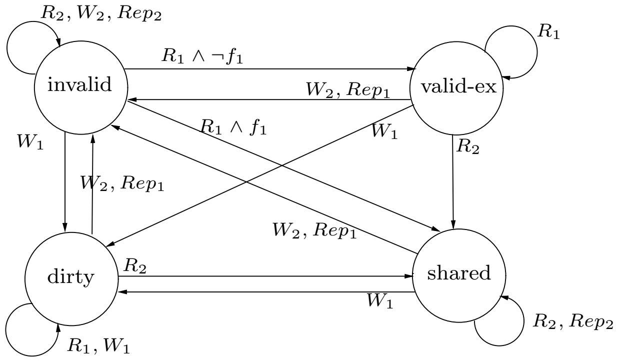

Fig. 1. Illinois protocol for 2 caches viewed from the perspective of cache C_1.

may have copies of the data; *dirty*, the cache has a modified copy of the data, i.e., the data in main memory are obsolete and the content of the other caches is not valid. The operations are R_i (read), W_i (write), and Rep_i (replace) for $i : 1, \ldots, k$=number of caches. The characteristic predicates f_i in $\mathcal{F}_{ill}$ is used by cache C_i to decide whether or not to move from *invalid* to *valid-exclusive*. Formally, f_i is defined as follows.

$$f_i(\langle q_1, q_2, \ldots, q_k \rangle) \text{ if and only if } \exists j \neq i \textit{ such that } q_j \neq \textit{invalid}$$

The possible transitions from the perspective of cache C_i are as follows.

Read Hit: no coherence action needs to be taken.

Read Miss: if there exists a cache C_j whose state is *dirty* (i.e. $f_i = true$), C_j supplies the missing block to C_i and updates the main memory. Both C_i and C_j end up in state *shared*. If there are *shared* or *valid-exclusive* copies in other caches (i.e. $f_i = true$), C_i gets the missing block from one of the caches and all caches with a copy end up in state *shared*. If there is no cached copy (i.e. $f_i = false$), C_i receives a *valid-exclusive* copy from main memory.

Write Hit: if C_i is in state *dirty*, no action is taken. If C_i is in state *valid-exclusive*, its state changes to *dirty* (note: no invalidation signal is needed). If C_i is in state *shared*, its state changes to *dirty* and all remote copies must be invalidated.

Write Miss: similar to *Read Miss*, except that all remote copies are invalidated and the state of C_i changes to *dirty*.

Replace: if C_i is in state *dirty*, the data are written back to memory.

For $k = 2$, the protocol from the perspective of cache C_1 is shown in Fig. 1.

Safety Properties. In this paper we limit ourselves to verification of *safety* property for data consistency [PD97]. As illustrated before, in the Illinois protocol the state *shared* indicates that a cache has a clean copy consistent with the memory and other caches copies, whereas the state *dirty* indicates that it has the latest and sole copy. Thus, in this example there are two possible sources of data inconsistency:

> INV_1: a *dirty* cache co-exists with one or more caches in state *shared*;
> INV_2: there are more than one *dirty* cache.

Thus, all global states that satisfy conditions INV_1 or INV_2 are *unsafe*. As mentioned before, we are interested in proving the protocol safe for every possible number of caches. For a fixed number of processors k and a given protocol P, let $\mathcal{I}(k)$ be the set of initial global states and $\mathcal{U}(k)$ be the set of unsafe global states. The *parameterized reachability problem* is defined as follows.

$$\exists\ k \geq 1.\ \exists\ G_1 \in \mathcal{I}(k).\ \exists\ G_2 \in \mathcal{U}(k) \text{ such that } G_1 \overset{*}{\to} G_2?$$

If the previous statement is true for a given k' than the protocol is not correct, i.e., an unsafe state may be reached for a system configuration with k' caches.

3 EFSMs for Parameterized Cache Coherence Protocols

In order to check parameterized safety properties we apply the following *abstraction*. Let Q be the set of cache states $q_1, \ldots, q_n$, then

> *we keep track only of the* number x_i *of processes in every state* $q_i \in Q$.

A global state G with k components (k=number of caches) is mapped to a tuple of *positive integers* with n components (n=number of cache states). This way, all *symmetric* global states are clustered together into a single representation. Via this abstraction, we cannot prove properties of individual caches like 'cache i and cache j cannot be in state *dirty* simultaneously'. However, we can still try to prove global properties like 'two different caches cannot be in state *dirty* simultaneously'. This is the kind of properties we are interested in to prove that the protocol will not give inconsistent (wrt. the semantics of states) results. The behavior of an *arbitrary* number of caches can be described *finitely* as a set of linear transformations describing the effect of the actions on the *counters* associated to the states in Q. For this purpose, we model the 'abstract protocol' as a single *Extended Finite State Machine* (EFSM) [CK97], i.e., a finite automaton with data variables (ranging over integers) associated to the locations and with guarded linear transformations associated to the transitions. Formally, let $\mathcal{M}_G$ be the global machine $\langle Q, \Sigma_G, \mathcal{F}, \delta_G \rangle$ associated to a protocol P, and let $Q = \{q_1, \ldots, q_n\}$. We model $\mathcal{M}_G$ as an EFSM with *only one location* and n data variables $\langle x_1, \ldots, x_n \rangle$ ranging over *positive integers*. For simplicity, we will always omit the location. The EFSM-states are tuples of natural numbers $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$, where c_i denotes the number of caches with state $q_i \in Q$ during a run of $\mathcal{M}_G$. The transitions are represented via a collection of *guarded linear transformations* defined over the variables $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{x}' = \langle x'_1, \ldots, x'_n \rangle$ and having the following form

$$G(\mathbf{x})\ \to\ T(\mathbf{x}, \mathbf{x}'),$$

where x_i and x'_i denote the number of caches in state q_i respectively before and after the occurrence of the transition. The guard $G(\mathbf{x})$ may be an arbitrary linear constraint over the variables $\mathbf{x}$. However, in this paper we limit ourselves to constraints defined as $D_1 \cdot \mathbf{x} \geq \mathbf{b} \ \wedge \ D_2 \cdot \mathbf{x} = \mathbf{c}$ where D_1 is an $n \times n$ matrix with $0, 1$ coefficients, D_2 is a diagonal $n \times n$-matrix with $0, 1$ coefficients, and $\mathbf{b}$ and $\mathbf{c}$ are vectors of integers. This type of guards allows us to handle both *local* and *global* conditions over the global states of $\mathcal{M}_G$. For instance, consider again the function $\mathcal{F}_{ill}$ of the Illinois protocol. Then, $f_i = \mathit{false}$ for some i can be expressed as $x_1 \geq 1, x_2 = 0, \ldots, x_n = 0$. For the sake of this paper, the transformation $T(\mathbf{x}, \mathbf{x}')$ is defined as $\mathbf{x}' = M \cdot \mathbf{x} + \mathbf{c}$ where M is an $n \times n$-matrix with unit vectors as columns (i.e., there is exactly one non-zero coefficient $= 1$ in each column). This way, we can represent the changes of states of the caches in the system (including the invalidation signals). Since the number of caches is an invariant of the system, we require the transformation to satisfy the condition $x'_1 + \ldots + x'_n = x_1 + \ldots + x_n$.

Remark 1. Let us call *additive constraint* a system of *linear inequalities* having the form $D \cdot \mathbf{x} \geq \mathbf{c}$ where D is a matrix with $0, 1$ coefficients, and $\mathbf{c}$ is a vector of positive integers. (Note: an additive constraint can be expressed as a conjunction of atomic formulas $x_{i_1} + \ldots + x_{i_k} \geq c$ where $x_{i_1}, \ldots, x_{i_k}$ are distinct variables from $\mathbf{x}$, and c is a positive integer.) When all guards of an EFSM are restricted to *additive* constraints, we obtain the subclass of *broadcast protocols* introduced in [EN98]. Thus, in broadcast protocols it is not possible to enforce *global* conditions that, e.g, require tests for zero (constants).

We show next some general patterns we use to model protocol actions via guarded transformations.

Internal action. A cache moves from state q_1 to state q_2 ($q_1 \neq q_2$): $x'_1 = x_1 - 1, x'_2 = x_2 + 1$ with the proviso that $x_1 \geq 1$ is part of $G(\mathbf{x})$.

Rendez-vous. Let us consider the case in which two caches synchronize on a signal. A cache C in state q_1 synchronizes with a cache C' in state q_3, the state of C changes to q_2, the state of C' changes to q_4 (all states are different). We model this transition as $x'_1 = x_1 - 1, x'_2 = x_2 + 1, x'_3 = x_3 - 1, x'_4 = x_4 + 1$, with the proviso that $x_1 \geq 1, x_3 \geq 1$ is part of $G(\mathbf{x})$.

Synchronization. All the caches in state $q_1, \ldots, q_m$ go to state q_i, e.g., for $i > m$. We model this transition as $x'_1 = 0, \ldots, x'_m = 0, x'_i = x_i + x_1 + \ldots + x_m$. This feature can be used, e.g., to model *bus invalidation signals.*

A run of an EFSM $\mathcal{E}$ is a (possibly infinite) sequence of EFSM-states $\mathbf{c_1}, \ldots, \mathbf{c_i} \ldots$ where $G_r(\mathbf{c_i}) \wedge T_r(\mathbf{c_i}, \mathbf{c_{i+1}}) = \mathit{true}$ for some transitions $G_r \to T_r$ in $\mathcal{E}$. We will denote the existence of a run from $\mathbf{c}$ to $\mathbf{c}'$ as $\mathbf{c} \xrightarrow{*} \mathbf{c}'$ Finally, we define a *predecessor* operator $\mathit{pre} : \mathcal{P}(\mathbb{N}^n) \rightsquigarrow \mathcal{P}(\mathbb{N}^n)$ over *sets* of EFSM-states as follows.

$$\mathit{pre}(S) = \{\mathbf{c} \mid \mathbf{c} \ \to \ \mathbf{c}', \ \mathbf{c}' \in S\}.$$

Here $\to$ indicates a one-step EFSM state transition.

$(r1)$ $dirty + shared + exclusive \geq 1 \ \ \rightarrow \ \ .$

$(r2)$ $invalid \geq 1, dirty = 0, shared = 0, exclusive = 0 \ \ \rightarrow$
$\quad invalid' = invalid - 1, exclusive' = exclusive + 1.$

$(r3)$ $invalid \geq 1, dirty \geq 1 \ \ \rightarrow$
$\quad invalid' = invalid - 1, dirty' = dirty - 1, shared' = shared + 2.$

$(r4)$ $invalid \geq 1, shared + exclusive \geq 1 \ \ \rightarrow$
$\quad invalid' = invalid - 1, shared' = shared + exclusive + 1, exclusive' = 0.$

$(r5)$ $dirty \geq 1 \ \ \rightarrow \ \ .$

$(r6)$ $exclusive \geq 1 \ \ \rightarrow \ \ exclusive' = exclusive - 1, dirty' = dirty + 1.$

$(r7)$ $shared \geq 1 \ \ \rightarrow$
$\quad shared' = 0, invalid' = invalid + shared - 1, dirty' = dirty + 1.$

$(r8)$ $invalid \geq 1 \ \ \rightarrow \ \ invalid' = invalid + exclusive + dirty + shared - 1,$
$\quad exclusive' = 0, shared' = 0, dirty' = 1.$

$(r9)$ $dirty \geq 1 \ \ \rightarrow \ \ dirty' = dirty - 1, invalid' = invalid + 1.$

$(r10)$ $shared \geq 1 \ \ \rightarrow \ \ shared' = shared - 1, invalid' = invalid + 1.$

$(r11)$ $exclusive \geq 1 \ \ \rightarrow \ \ exclusive' = exclusive - 1, invalid' = invalid + 1.$

Fig. 2. EFSM for the Illinois Protocol: all variables range over *positive* integers.

3.1 The EFSM for the Illinois Protocol

Let *invalid*, *dirty*, *shared*, and *exclusive* be variables ranging over *positive integers*. The EFSM for the Illinois protocol is shown in Fig. 2. For simplicity, we omit the location and all equalities of the form $x_i' = x_i$. Furthermore, in a rule like '$G \rightarrow$ ' all variables remain unchanged. Rule $r1$ of Fig. 2 represents *read hit* events: since no coherence action is needed, the only precondition is that there exists at least one cache in a valid state, i.e., $dirty + shared + exclusive \geq 1$. Rules $r2 - r4$ correspond to *read miss* events where the global predicate $\mathcal{F}_{ill}$ is expressed via guards containing tests for zero. Specifically, rule $r2$ represents a read miss such that $f_i = false$ for some i, i.e., one cache can move to *valid-exclusive*. The case in which a cache copies its content from a *dirty* cache and the two caches move simultaneously to *shared* is defined via rule $r3$. Rule $r4$ applies whenever the block is copied from a cache in *shared* or *valid-exclusive* state. Rules $r5 - r7$ model *write hits*. Specifically, rule $r5$ models a write in state *dirty* (no action is taken). Rule $r6$ models a write in state *valid-exclusive* where the state changes to *dirty* without bus invalidation signal. Rule $r7$ models a write in state *shared* where the copies in all other caches are invalidated. Note that, in rule $r7$ we implicitly assume that whenever $share \geq 1$ then $dirty = 0$. We will (automatically) prove that this is an invariant of the protocol in Section 5. Rule $r8$ corresponds to a *write miss*: one cache moves to *dirty* and the copies in the other caches are invalidated. Finally, rules $r9 - r11$ model *replacement* events. If the cache is in one of the states *dirty*, *shared* or *exclusive* its state changes to *invalid*.

4 Protocol Validation = EFSM Invariant Checking

Let P be a protocol with global machine $\mathcal{M}_G$ and states $Q = \{q_1, \ldots, q_n\}$. Given a global state G, we define $\#G$ as the tuple of natural numbers $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$ where c_i=*number of caches in* G *with state* $= q_i$. Now, let $\mathcal{E}_P$ be the EFSM in which a state $\mathbf{c}$ represents the set of global states $\{G \mid \#G = \mathbf{c}\}$, and whose transitions are obtained according to Section 3. The following proposition relates runs in $\mathcal{M}_G$ and runs in $\mathcal{E}_P$.

Proposition 1 (Adequacy of the Encoding). *Let $\mathcal{M}_G$ and $\mathcal{E}_P$ be defined as above. Then, $\mathbf{c} \overset{*}{\to} \mathbf{d}$ in $\mathcal{E}_P$ if and only if there exist two global states G and G', such that $\#G = \mathbf{c}$, $\#G' = \mathbf{d}$ and $G \overset{*}{\to} G'$ in $\mathcal{M}_G$.*

The previous property allows us to reduce the *parameterized reachability* problem for coherence protocols to a *reachability* problem for the corresponding EFSMs. Our approach to attack the second problem is based on the following points.

Symbolic State Representation. In order to represent concisely (possibly infinite) sets of global states independently from the number of caches in the system, we use *linear arithmetic constraints*=systems of linear inequalities as a symbolic representation of sets of EFSM-states. This class of constraints is powerful enough to express *initial* and *unsafe (target)* sets of states for the verification problems we are interested in. For instance, the set of unsafe states of the Illinois protocol where at least 1 cache is in state *shared* and *at least* 1 cache is in state *dirty* can be represented finitely as the constraint $x_{shared} \geq 1 \wedge x_{dirty} \geq 1$. This is a crucial aspect in order to attack the *parameterized reachability problem.* In the rest of the paper we will use the lower-case letters $\varphi, \psi, \ldots$ to denote constraints and the upper-case letter $\Psi, \Phi, \ldots$ to denote *sets* (disjunctions) of constraints. Following [ACJT96], the *denotation* of a constraint φ is defined as $[\![\varphi]\!] = \{\mathbf{t} \mid \mathbf{t} \in \mathbb{N}^n$ satisfies $\varphi\}$. The definition is extended to sets in the natural way. Furthermore, we say that a constraint ψ *entails* a constraint φ, written $\varphi \sqsubseteq \psi$, iff $[\![\psi]\!] \subseteq [\![\varphi]\!]$. We define a *symbolic predecessor operator* **sym_pre** over sets of constraints such that $[\![\mathbf{sym\_pre}(\Phi)]\!] = pre([\![\Phi]\!])$ (*pre* is defined in Section 3). The operator is defined via the satisfiability test and variable elimination over $\mathbb{N}$ (the domain of interpretation of constraints) [BGP97, DP99]. Formally, for a given constraint $\varphi(\mathbf{x}')$ with variables over $\mathbf{x}'$, **sym_pre** is defined as follows

$$\mathbf{sym\_pre}(\varphi(\mathbf{x}')) = \bigcup_{i \in I} \;\{\; \exists\, \mathbf{x}'.\; G_i(\mathbf{x}) \;\wedge\; T_i(\mathbf{x}, \mathbf{x}') \;\wedge\; \varphi(\mathbf{x}') \;\}$$

where $\mathbf{x}$ and $\mathbf{x}'$ range over $\mathbb{N}$, and $G_i(\mathbf{x}) \to T_i(\mathbf{x}, \mathbf{x}')$ is an EFSM transition rule for $i \in I$ (I=index set).

Backward Reachability. We apply a variation of the *backward reachability* algorithm of [ACJT96], where all operations on sets of states are lifted to the constraint-level. The reason we adopt *backward reachability* is due to the result proved in [EFM99]: in contrast to *forward reachability*, the algorithm of [ACJT96] always terminates whenever the input EFSM is a *broadcast protocol* [EN98] and the set of unsafe states is upward-closed. A set $S \subseteq \mathbb{N}^k$ of states is upward-closed whenever for all tuples $\mathbf{t} \in S$: if $\mathbf{t}'$ is greater equal than $\mathbf{t}$ w.r.t. the componentwise ordering of tuples, then $\mathbf{t}' \in S$. As an example, the denotation of the constraint $x_{shared} \geq 1 \wedge x_{dirty} \geq 1$ (the variables $x_{invalid}$ and $x_{valid-ex}$ are implicitly ≥ 0) is an upward-closed set over $\mathbb{N}^4$. As shown in [DEP99], the result of [EFM99] implies that the symbolic reachability algorithm using *integer constraints* to represent sets of states always terminates on inputs consisting of a broadcast protocol and of constraints that represents upward-closed sets of unsafe states.

Relaxation of Constraint Operations. In order to reduce the complexity of the manipulation of arithmetic constraints, we follow techniques used, e.g., in integer programming and program analysis. Every time we need to solve a system of inequalities $A \cdot \mathbf{x} \leq \mathbf{b}$ we 'relax' the condition that $\mathbf{x}$ must be a vector of positive *integers* and look for a *real* solution of the corresponding linear problem. We apply the *relaxation* to the operations over constraints, i.e., we interpret the *satisfiability* test, *variable elimination*, and *entailment* test (needed to implement the symbolic backward reachability algorithm) over the domain of *reals*. The relaxation allows us to exploit efficient (polynomial) operations over the reals in contrast to potentially exponential operations over the integers. Note that this abstraction is applied *only during* the analysis and not at the semantic level of EFSMs. As we will discuss later, in many cases this method does not lose precision wrt. an analysis over the integers. Formally, given a constraint φ, we define $[\![\varphi]\!]_{\mathbb{R}}$ as the set of real solutions $\{\mathbf{t} \in \mathbb{R}_+ \mid \mathbf{t}\ satisfies\ \varphi\ \}$. The entailment relation over $\mathbb{R}_+$ is defined then as $\varphi \sqsubseteq_{\mathbb{R}} \psi$ if and only if $[\![\psi]\!]_{\mathbb{R}} \subseteq [\![\varphi]\!]_{\mathbb{R}}$. When we apply the above relaxation to the symbolic predecessor operator, we obtain the new operator $\mathbf{sym\_pre}_{\mathbb{R}}$ defined as follows

$$\mathbf{sym\_pre}_{\mathbb{R}}(\varphi(\mathbf{x}')) = \ \bigcup_{i \in I} \ \{ \ \exists\ \mathbf{x}'.\ G_i(\mathbf{x}) \ \wedge \ T_i(\mathbf{x}, \mathbf{x}') \ \wedge \ \varphi(\mathbf{x}') \ \},$$

where $\mathbf{x}$ and $\mathbf{x}'$ range now over *positive real numbers*, and $\exists_{\mathbb{R}} x.F \ \equiv \ \exists x \in \mathbb{R}_+.F$ The symbolic reachability algorithm we obtained is shown in Fig. 3. Note that this is the algorithm for backward reachability implemented in existing symbolic model checkers for hybrid and concurrent systems like HyTech [HHW97] and DMC [DP99]. Each step of the procedure Symb-Reach-over-$\mathbb{R}$ involves only polynomial time cost operations. In fact, $\mathbf{sym\_pre}_{\mathbb{R}}$ can be implemented using satisfiability test over $\mathbb{R}$ (e.g. using the simplex method, 'polynomial' in practical examples) and using Fourier-Motzkin variable elimination (for a fixed number of variables, polynomial in the size of the input constraints). Furthermore, the entailment test $\varphi \sqsubseteq_{\mathbb{R}} \psi$ can be tested in polynomial time. In fact, $\phi \ \sqsubseteq_{\mathbb{R}} \ (\psi_1 \wedge \psi_2)$ holds if and only if $\phi \ \wedge \ \neg\psi_1$ and $\phi \ \wedge \neg\psi_2$ are not satisfiable. Thus, the entailment test reduces to a linear (in the size of the input constraints) number of

```
Proc Symb-Reach-over-$\mathbb{R}$($\Phi_o, \Phi_f$ : sets of constraints)
  set $\Phi := \Phi_f$ and $\Psi := \emptyset$;
  while $\Phi \neq \emptyset$ do
    choose $\varphi \in \Phi$ and set $\Phi := \Phi \setminus \{\varphi\}$;
    if there are no $\psi \in \Psi$ s.t. $\psi \sqsubseteq_{\mathbb{R}} \varphi$ then
        set $\Psi := \Psi \cup \{\varphi\}$ and $\Phi := \Phi \cup \mathbf{sym\_pre}_{\mathbb{R}}(\varphi)$;
  if $sat_{\mathbb{R}}(\Phi_o \wedge \Psi)$ then return '$\Phi_f$ is reachable from $\Phi_o$';
  else return '$\Phi_f$ is not reachable from $\Phi_o$'.
```

Fig. 3. Symbolic reachability.

satisfiability tests. In contrast, the cost of executing the same operations over $\mathbb{N}$ may be exponential. For instance, in [DEP99] we have shown (via a reduction from the Hitting Set problem) that checking $\varphi \sqsubseteq \psi$ (over $\mathbb{N}$) is already co-NP hard whenever φ and ψ are two instances of the *additive* constraints of Remark 1, Section 3 (i.e., constraints without equalities).

As in other verification methods for infinite-state systems (e.g. for hybrid systems [HHW97], FIFO systems [BW98], and parameterized concurrent programs [ABJN99]), the algorithm is not guaranteed to terminate on every input. This is the price we have to pay in order to model realistic examples. We give next sufficient conditions for the *theoretical* termination of Symb-Reach-over-$\mathbb{R}$. We postpone the evaluation of its *practical* termination to Section 5.

Sufficient Conditions for Termination. As for its companion algorithm defined over the domain of integers, the procedure Symb-Reach-over-$\mathbb{R}$ always terminates, returning exact results, if both the guards of the input EFSM and the unsafe states are expressed via the additive constraints of Remark 1, Section 3. This result proves that, when applied to broadcast protocols, the algorithm of [ACJT96] formulated over constraints is *robust under the relaxation integer-reals* of the constraint operations (*robust*=it always terminates and solves the reachability problem taken into consideration). Formally, we have the following result.

Theorem 1. *Given a broadcast protocol P, Symb-Reach-over-*$\mathbb{R}(\Phi_o, \Phi_f)$ solves *the reachability problem '* $\exists \mathbf{c_0} \in [\![\Phi_0]\!], \exists \mathbf{c_1} \in [\![\Phi_f]\!]$ *such that* $\mathbf{c_0} \xrightarrow{*} \mathbf{c_1}$*', whenever* Φ_0 *is a set of* additive *constraints (possibly extended with conjunctions of equalities of the form* $x_i = c_i$, $c_i \in \mathbb{N}$*), and* Φ_f *is a set of* additive *constraints.*

Sketch of the proof. Following the methodology of [AJ99, FS98], we need to prove the following lemmas: (1) given an additive constraint φ, we can effectively compute $\mathbf{sym\_pre}_{\mathbb{R}}(\varphi)$; (2) the class of additive constraints is closed under application of $\mathbf{sym\_pre}_{\mathbb{R}}$; and, finally, (3) the class of additive constraints equipped with the order $\sqsubseteq_{\mathbb{R}}$ is a *well-quasi ordering*, i.e., there exist no infinite chains of additive constraints $\varphi_1 \ldots \varphi_i \ldots$ such that $\varphi_j \not\sqsubseteq_{\mathbb{R}} \varphi_i$ for all $j < i$. Point (1) follows from our definition of $\mathbf{sym\_pre}_{\mathbb{R}}$, whereas point (2) and (3) are formally proved in [Del00]. For instance, the proof of lemma (2) is based on the following observation. Let $\varphi(\mathbf{x}')$ be an additive constraint and r be a transition $G(\mathbf{x}) \to T(\mathbf{x}, \mathbf{x}')$. Then, we can compute $\mathbf{sym\_pre}_{\mathbb{R}}(\varphi)$ (restricted to r) by replacing all 'primed variables' in φ with the right-hand side of the transformation $T(\mathbf{x}, \mathbf{x}')$, and by conjoining the resulting constraint with the guard $G(\mathbf{x})$. The special form of $T(\mathbf{x}, \mathbf{x}')$ (each variable occurs only once in the right-hand side of assignments) ensures that the resulting constraint is still an additive constraint. As a corollary of lemma (2), it follows that computing symbolically the predecessor of an additive constraint φ over $\mathbb{R}$ and over $\mathbb{N}$ gives the same results (both computations amounts to a replacement by equals). In other words, $\mathbf{sym\_pre}_{\mathbb{R}}$ gives accurate results, i.e., $[\![\mathbf{sym\_pre}_{\mathbb{R}}(\Phi)]\!] = [\![\mathbf{sym\_pre}(\Phi)]\!] = pre([\![\Phi]\!])$. (Note: $[\![\cdot]\!]$ denotes *integer* solutions). $\square$

To our knowledge, this result was not considered in previous works on well structured system [ACJT96, AJ99, FS98]. In the rest of the paper we will discuss some preliminary experimental results.

5 Experimental Results

We have applied HyTech and DMC to automatically verify safety properties for the MESI, Berkeley RISC, Illinois, Xerox PARC Dragon and DEC Firefly protocols [Han93]. The guards we need to model the Dragon and Firefly protocols are more complicated than those of the Illinois protocol. The results of the analysis are shown in Fig. 4. For instance, in the Illinois protocol the parameterized initial configuration is expressed as $\Phi_o = invalid \geq 1,\ exclusive = 0,\ dirty = 0,\ shared = 0$. Similarly, we can represent the potentially *unsafe* states described in Section 2 as follows: $\Phi_1 = invalid \geq 0,\ exclusive \geq 0,\ dirty \geq 1,\ shared \geq 1$ (property INV_1) and $\Phi_2 = invalid \geq 0,\ exclusive \geq 0,\ dirty \geq 2,\ shared \geq 0$ (property INV_2). We automatically checked both properties using HyTech and DMC (without need of accelerations) as specified in Fig. 4. HyTech execution times are often better (HyTech is based on Halbwachs' efficient *polyhedra* library [Hal93]). However, the HyTech built-in command *reach backward* we use for the analysis does not terminate in two cases (see table). DMC terminates on all examples thanks to a set of built-in *accelerations* [DP99]. Similar techniques (e.g. extrapolation) are described in [HH95, LHR97] but they are not provided by the current version of HyTech. We have tried other experiments using HyTech: *forward* analysis using *parameter* variables (to represent the initial configurations) does not terminate in several examples; approximations based on the *convex hull* (using the built-in *hull* operator applied to intermediate collections of states) returned no interesting results. We have also experimented other type of abstractions. Specifically, we have analyzed the EFSMs obtained by weakening the guards of the original descriptions (e.g. turning tests for zero into inequalities) so as to obtain EFSMs for which our algorithm always terminates. As shown by the 'question marks' in the results for *Abstract Illinois* in Fig. 4, with this abstraction we find *errors* that are not present in the original protocol (in fact, the resulting reachable states are a super-set of those of the Illinois protocol).

Protocol	Unsafe-GS	HyTech-ET[1]	HyTech-NS	DMC-ET[2]	DMC-NS	Safe?
Mesi	D>2	0.77s	3	1.0s	3	yes
	D>1,S>1	0.66s	2	0.6s	2	yes
Berkeley RISC	D>2	0.52s	1	0.3s	1	yes
	D>1,S>1	0.94s	3	1.5s	3	yes
University of Illinois	D>2	1.06s	3	2.0s	3	yes
	D>1,S>1	2.32s	4	10.3s	4	yes
DEC Firefly	D>2	↑	-	28.2s	7	yes
	D>1,S>1	3.01s	4	11.4s	4	yes
Xerox PARC Dragon	D>2	↑	-	84.2s	6	yes
	D>1,S_c >1	5.30s	5	25.1s	5	yes
	D>1,S_d >1	5.26s	5	25s	5	yes
Abstract Illinois	D>2	2.86s	5	16.9s	5	?
	D>1,S>1	8.14s	7	96.3s	7	?

[1] on a Sun-SPARCstation-5 OS 5.6 [2] on a Pentium 133 Linux 2.0.32

Fig. 4. ET=Execution Time; NS=No. Steps (↑=diverges). Unsafe-Global States: D=Dirty, S=Shared, S_c=Shared-Clean, S_d=Shared-Dirty. Abstracted Illinois is obtained by weakening the guards of Illinois.

6 Related Works

Our approach is inspired by the recent work of Emerson and Namjoshi on Parameterized Model Checking [EN96, EN98b], and broadcast protocols [EN98]. As discussed in the paper, broadcast protocols are not general enough to model the *global* conditions required by the protocol we have validated in this paper. The verification technique proposed in [EN98] is an extension of the *coverability graph* for Petri Nets (forward reachability). For broadcast protocols, this construction is not guaranteed to terminate [EFM99]. In contrast, backward reachability always terminates when the target set of states is upward-closed [EFM99]. In [DEP99], the author in collaboration with Esparza and Podelski proposes efficient data structures for representing integer constraints for the verification of broadcast protocols. There exist specialized symbolic state exploration techniques for the analysis of *parameterized* coherence protocols. In [PD95], Pong and Dubois propose the *symbolic state model* (SSM) for the representation of the state-space and the verification of protocols. Specifically, they apply an *abstraction* and represent sets of global states via *repetition* operators to indicate 0,1, or multiple caches in a particular state. In our EFSM-model we keep track of the *exact* number of processes in each state. SSM verification method is based on a *forward* exploration with *ad hoc* expansion and aggregation rules. In [ID99], Norris Ip and Dill have incorporated the repetition operators in Murφ. Murφ automatically checks the soundness of the abstraction based on the repetition operators, verifies the correctness of an abstract state graph of a fixed size using *on_the_fly state enumeration*, and, finally, tries to generalize the results for systems with larger (unbounded) sizes. In contrast, our method is based on general purpose techniques (backward reachability and constraints to represent sets of states) that have been successfully applied to the verification of timed, hybrid and concurrent systems (see e.g. [HHW97, BGP97, DP99]). Being specialized to coherence protocols, SSM can also detect stale write-backs and livelocks. The verification of this type of properties using our method is part of our future works. We are not aware of other applications of infinite-state symbolic model checkers based on arithmetical domains to verification of parameterized cache coherence protocols. Several approaches exist to attack the verification problem of parameterized *concurrent systems.* In [GS92], German and Sistla define an automatic method for verification of parameterized *asynchronous* systems (where processes are model in CCS-style). However, methods that handle *global conditions* like ours (e.g. [ABJN99]) are often semi-algorithms, i.e., they do not guarantee the termination of the analysis. Other methods based on *regular languages* have been proposed in [ABJN99, CGJ97]. Among semi-automatic methods that require the construction of abstractions and invariants we mention [BCG89, CGJ97, HQR99, McM99]. Automated generation of invariants has been studied e.g. in [CGJ97, LHR97]. Automated generation of abstract transition graphs for infinite-state systems has been studied in [GS97]. Symmetry reductions for parameterized systems have been considered, e.g., in [ID99, McM99, PD95].

Finally, in [Del00] the author shows that the method presented in this paper (backward reachability, constraint-based representation, relaxation) can be used as a general methodology to verify properties of parameterized *synchronous systems.*

7 Conclusions

We have proposed a new method for the verification of coherence protocols for any number of processors in the system. We have applied our methods to successfully verify safety properties of several protocols taken from the literature [AB86, Han93, PD95]. This result is obtained using technology originally developed for the verification of hybrid and concurrent systems, namely HyTech [HHW97] and DMC [DP99]. In our approach we propose the following *abstractions.* We 'count' the number of caches in every possible protocol state, so that we get an integer system out of a parameterized protocol; we relax the constraint operations needed to implement the symbolic backard reachability algorithm for the resulting integer system. The abstraction based on the relaxation often gives *accurate* results (e.g. when intermediate results are additive constraints) and allows us to prove all the properties of our examples we were interested in. As discussed in Section 5, with other types of approximation techniques we might abstract away crucial properties of the original protocols. As future works, we plan to extend our method to other classes of coherence protocols (e.g. *directory-based* [Han93]), and properties (e.g., *livelocks*), and to study techniques to provide *error traces* for properties whose verification fails.

Acknowledgements. The author would like to thank Parosh Aziz Abdulla, Howard Wong-Toi, Kedar Namjoshi, Javier Esparza and the anonymous referees for helpful references, suggestions, and encouragement.

References

[ACJT96] P. A. Abdulla, K. Cerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. 10th IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.

[ABJN99] P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling Global Conditions in Parameterized System Verification. In *Proc. 11th Int. Conf. on Computer Aided Verification (CAV'99)*, LNCS 1633, pages 134–145, 1999.

[AJ99] P. A. Abdulla, and B. Jonsson. Ensuring Completeness of Symbolic Verification Methods for Infinite-State Systems. To appear in *Theoretical Computer Science*, 1999.

[AB86] P. A. Archibald and J. Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. *ACM Transactions on Computer Systems* 4(4): 273–298. 1986.

[BGP97] T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite-state Systems using Presburger Arithmetics. In *Proc. 9th Conf. on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 400–411, 1997.

[BW98] B. Boigelot and P. Wolper. Verifying Systems with Infinite but Regular State Space. In *Proc. 10th Conf. on Computer Aided Verification (CAV'98)*, LNCS 1427, pages 88–97, 1998.

[BCG89] M. C. Browne, E. M. Clarke, O. Grumberg. Reasoning about Networks with Many Identical Finite State Processes. *Information and Computation* 81(1): 13–31, 1989.

[CK97] K.-T. Cheng and A. S. Krishnakumar. Automatic Generation of Functional Vectors Using the Extended Finite State Machine Model. *ACM Transactions on Design Automation of Electronic Systems* 1(1):57–79, 1996.

[CGH+93] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+cache coherence protocol. In *Proc. 11th Int. Symp. on Computer Hardware Description Languages and their Applications*, 1993.

[CGJ97] E. Clarke, O. Grumberg, and S. Jha. Verifying Parameterized Networks. *TOPLAS* 19(5): 726–750, 1997.

[Del00] G. Delzanno. On Efficient Data Structures for the Verification of Parameterized Synchronous Systems. Tech. Rep. DISI-00-03, Dip. di Informatica e Scienze dell'Informazione, Università di Genova, January 2000.

[DEP99] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based Analysis of Broadcast Protocols. In *Proc. Annual Conf. of the European Association for Computer Science Logic (CSL'99)*, LNCS 1683, pag. 50–66, 1999.

[DP99] G. Delzanno and A. Podelski. Model Checking in CLP. In *Proc. 5th Int. Con. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, LNCS 1579, pages 223–239, 1999.

[EN96] E. A. Emerson and K. S. Namjoshi. Automatic Verification of Parameterized Synchronous Systems. In *Proc. 8th Conf. on Computer Aided Verification (CAV'96)*, LNCS 1102, pages 87–98, 1996.

[EN98] E. A. Emerson and K. S. Namjoshi. On Model Checking for Nondeterministic Infinite-state Systems. In *Proc. of the 13th Annual Symp. on Logic in Computer Science (LICS '98)*, pages 70–80, 1998.

[EN98b] E. A. Emerson and K. S. Namjoshi. Verification of Parameterized Bus Arbitration Protocol. In *Proc. 10th Conf. on Computer Aided Verification (CAV'98)*, *LNCS* 1427, pages 452–463, 1998.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. 14th Annual Symp. on Logic in Computer Science (LICS'99)*, pages 352–359, 1999.

[FS98] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! Tech. Rep. LSV-98-4, Lab. Spécification et Vérification, ENS de Cachan, April 1998. To appear in *Theoretical Computer Science*.

[GS92] S. M. German, A. P. Sistla. Reasoning about Systems with Many Processes. *JACM* 39(3): 675–735 (1992)

[GS97] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Proc. 9th Conf. on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 72–83, 1997.

[Hal93] N. Halbwachs. Delay Analysis in Synchronous Programs. In *Proc.5th Conf. on Computer-Aided Verification (CAV'93)*, LNCS 697, pages 333–346, 1993.

[Han93] J. Handy. The Cache Memory Book. Academic Press, 1993.

[HH95] T. A. Henzinger, P.-H. Ho. A Note on Abstract-interpretation Strategies for Hybrid Automata. In *Proceedings of Hybrid Systems II*, LNCS 999, pages 252–264, 1995.

[HHW97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a Model Checker for Hybrid Systems. In *Proc. 9th Conf. on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 460–463, 1997.

[HQR99] T. A. Henzinger, S. Qadeer, S. K. Rajamani. Verifying Sequential Consistency on Shared-Memory Multiprocessor Systems. In *Proc. 11th Int. Conf. on Computer Aided Verification (CAV'99)*, LNCS 1633, pages 301–315, 1999.

[ID99] C. Norris Ip and D. L. Dill. Verifying Systems with Replicated Components in Murphi. *Formal Methods in System Design*, 14(3): 273–310, 1999.

[LHR97] D. Lesens and N. Halbwachs and P. Raymond. Automatic Verification of Parameterized Linear Networks of Processes. In *Proc. 24th ACM Symposium on Principles of Programming Languages (POPL'97)*, pages 346–357, 1997.

[McM99] K. L. McMillan. Verification of Infinite State Systems by Compositional Model Checking. In *Proc. Conf. Correct Hardware Design and Verification Methods (CHARME'99)*, LNCS 1703, pages 219–234, 1999.

[MS91] K. L. McMillan and J. Schwalbe. Formal Verification of the Gigamax Cache Consistency Protocol. In *Proc. Int. Symp. on Shared Memory Multiprocessors*, pages 242–251, 1991.

[PP84] M. S. Papamarcos, J. H. Patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *Proc. Int. Symp. on Computer Architecture (ISCA 84)*, pages 348–354, 1984.

[PD95] F. Pong, M. Dubois. A New Approach for the Verification of Cache Coherence Protocols. *IEEE Transactions on Parallel and Distributed Systems* 6(8), 1995.

[PD97] F. Pong, M. Dubois. Verification Techniques for Cache Coherence Protocols. *ACM Computing Surveys* 29(1): 82–126, 1997.