

# View Abstraction – A Tutorial\*

Parosh A. Abdulla<sup>1</sup>, Fréderic Haziza<sup>2</sup>, and Lukáš Holík<sup>3</sup>

<sup>1</sup> Uppsala University

[parosh@it.uu.se](mailto:parosh@it.uu.se)

<sup>2</sup> Uppsala University

[frederic.haziza@it.uu.se](mailto:frederic.haziza@it.uu.se)

<sup>3</sup> Brno University of Technology

[holik@fit.vutbr.cz](mailto:holik@fit.vutbr.cz)

---

## Abstract

---

We consider *parameterized verification*, i.e., proving correctness of a system with an unbounded number of processes. We describe the method of *view abstraction* whose aim is to provide a *small model property*, i.e., showing correctness by only inspecting instances of the system consisting of a small fixed number of processes. We illustrate the method through an application to the classical Burns' mutual exclusion protocol.

**1998 ACM Subject Classification** D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

**Keywords and phrases** program verification, model checking, parameterized systems

**Digital Object Identifier** 10.4230/OASIcs.SynCoP.2015.1

**Category** Invited Paper

## 1 Introduction

The behavior of many types of systems can be described using one or more *parameters* such as the number of processes, or the sizes of the data structures that the system uses. The goal of *parameterized verification* is to prove (or refute) the correctness of the system for all values of the parameters. There are numerous applications where parameterized systems arise naturally:

- *Number of processes.* In a mutual exclusion protocol, an arbitrary number of processes may participate in a given session of the protocol. In a cache coherence protocol, an arbitrary number of threads may share a cache line. In a Petri net, there is no bound on the number of tokens that are generated during a run of the net.
- *Sizes of data structures.* The behaviors of recursive programs can be captured using unbounded stacks [18]. Data link protocols can be modeled by processes communicating through unbounded FIFO queues [8]. The latter have also recently been used to encode the behavior of programs running on weak memory models such TSO, PSO, and POWER [12, 2, 28].
- *Multiple parameters.* Timed Petri Nets (TPN) [10] extend the model of Petri nets by equipping each token with a real-valued clock. A TPN induces a system that is infinite in two dimensions. More precisely, a run of a TPN may generate an unbounded number

---

\* This work was partially supported by UPMARC, The Uppsala Programming for Multicore Architectures Research Center.

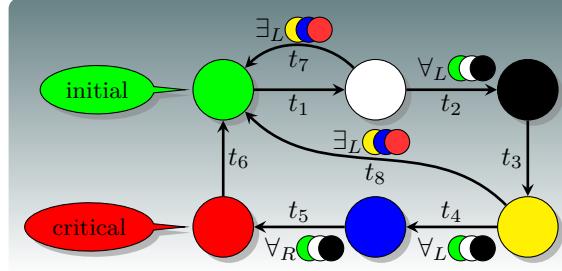
of tokens each of which is a real-valued clock. The implementations of concurrent stacks, queues, and sets are infinite in three dimensions [5]. First, an unbounded number of threads may operate on the data structure. Furthermore, there is no bound on the size of the data structure. Finally, the values stored inside the data structure are usually fetched from an infinite domain such as the set of integers.

In this tutorial, we concentrate on systems where the parameterization arises due to the number of processes. This class of systems can itself be divided into several subclasses depending on the following three parameters.

- *Processes.* The processes may be finite-state or infinite-state. Even in the case of finite-state processes, the state space of the system is unbounded. This is true since the state space contains all states we get as we vary the parameter (size of the system). The individual processes may be infinite-state since they may operate on variables ranging over infinite domains (e.g., the natural numbers). In such a case we get a state space that is infinite in two dimensions.
- *Topology.* On the one hand, the system may consist of a set of processes without any structure. On the other hand, the system topology may have a certain pattern. For instance, the processes may be organized as a linear array. Then, a process may refer to its left/right neighbors, or to all the processes to its left/right. The processes may also be organized in a ring, tree, or a general graph.
- *Communication Primitives.* A simple form of communication is when two processes perform a *rendezvous* which involves both processes changing state simultaneously. Another form of communication is through *shared variables* that can be read from and written to by all/some processes in the system. We may have *broadcast transitions* where an arbitrary number of processes change state simultaneously. Furthermore, the transitions of a process may be conditioned by *global conditions*. An example of a (universal) global condition, in a system with linear topology, is that “all processes to the left of a given process  $i$  should satisfy a property  $\phi$ ”. In this case, process  $i$  is allowed to perform the transition only in the case where all processes with indices  $j < i$  satisfy  $\phi$ .

In this paper, we consider a class of systems where we have finite-state processes that are organized in a linear array. The processes communicate through global transitions. For such systems, we consider the verification of *safety properties*. Intuitively, a safety property states that nothing bad will happen during the execution of the system. For a mutual exclusion protocol, a typical safety property is that no two processes should be in their critical sections at the same time. Checking a given safety property reduces to checking the reachability of a set of *bad* configurations, namely those that violate the property.

The work of this tutorial is based on the ideas presented in [4] that introduces *view abstraction*. View abstraction is inspired by strong empirical evidence that parameterized systems often enjoy a *small model property*. More precisely, it analyzes only a small number of processes (rather than the whole family) and shows that this is sufficient to capture the (un)reachability of bad configurations. On the one hand, bad configurations can often be characterized by minimal conditions that are possible to specify through a fixed number of *witness* processes. For instance, in a mutual exclusion protocol, a bad configuration contains *two* processes in their critical sections; and in a cache coherence protocol, a bad configuration contains *two* cache lines in their *exclusive* states. In both cases, having the two witnesses is sufficient to make the configuration bad (regardless of the actual size of the configuration). On the other hand, it is usually the case that such bad patterns (if existing) appear already in small instances of the system. View abstraction shows also that it is often the case that correctness can be established by only inspecting a small number of processes. We illustrate the method through an application to the classical Burns’ mutual exclusion protocol.



**Figure 1** A process in Burns' Protocol.

## Related Work.

*Regular model checking* [24, 15] performs parameterized verification by encoding the set of configurations using finite-state automata. The method has been augmented with techniques such as widening [13, 29], abstraction [14], and acceleration [9]. All these works rely on computing the transitive closure of transducers or on iterating them on regular languages.

There are numerous techniques less general than regular model checking, but that are lighter and more dedicated to the problem of parameterized verification. The idea of *counter abstraction* is to keep track of the number of processes which satisfy a certain property [22, 16, 17, 27]. In general, counter abstraction is designed for systems with unstructured or clique architectures, but may be used for systems with other topologies too.

Several works reduce parameterized verification to the verification of finite-state models. Among these, the *invisible invariants* method [11, 26] and the work of [25] exploit cut-off properties to check invariants for mutual exclusion protocols.

*Monotonic abstraction* [6, 7, 30] combines regular model checking with abstraction in order to produce systems that have monotonic behaviors wrt. a well quasi-ordering on the state-space. The method of [21, 20] and its reformulated, generic version of [19] come with a complete method for well-quasi ordered systems which is an alternative to backward reachability analysis based on a forward exploration.

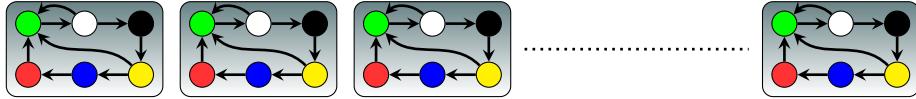
Parameterized systems whose behaviors are conditioned by time or data constraints are described in [3, 1].

## 2 Model

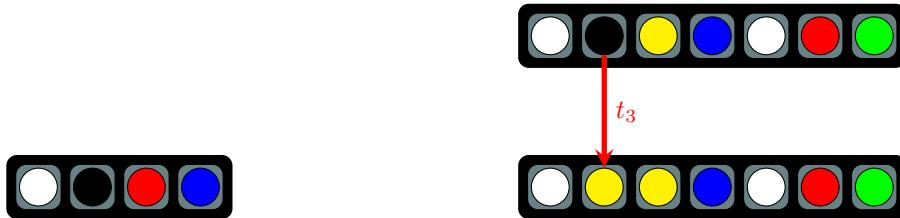
We consider parameterized systems where the processes are modeled as finite-state automata arranged in a linear array. The processes may perform local or global (existential or universal) transitions. We illustrate our model through the classical Burns' protocol.

### 2.1 Processes

A process in Burns' protocol, depicted in Fig. 1, is defined by a finite-state automaton. The automaton has six states, namely  $\bullet$ ,  $\circlearrowleft$ ,  $\bullet$ ,  $\bullet$ ,  $\bullet$ , and  $\bullet$ . The process starts its execution from the *initial* state  $\bullet$ , and tries to reach its *critical* section  $\bullet$ . The automaton contains three types of transitions. From the state  $\bullet$ , the process can perform the *local* transition  $t_1$  in which it changes state to  $\circlearrowleft$  regardless of the states of the other processes. From the state  $\circlearrowleft$ , the process can perform the *existential global* transition  $t_7$  in which it changes state to  $\bullet$  provided that there *exists* a process to its *left* (and hence the notation  $\exists_L$ ) whose state is either  $\bullet$ ,  $\bullet$ , or  $\bullet$ . From the state  $\circlearrowleft$ , the process can also perform the *universal*



■ **Figure 2** Parameterized Burns' Protocol.



■ **Figure 3** A configuration.

■ **Figure 4** A local transition.

*global* transition  $t_2$  in which it changes state to  $\bullet$  provided that the states of *all* processes to its *left* (and hence the notation  $\forall_L$ ) are either  $\text{green}$ ,  $\text{white}$ , or  $\bullet$ . A process starts its execution from (state)  $\text{green}$  and can immediately cross to  $\text{white}$ . At  $\text{white}$  it looks left and performs a test where it checks whether all processes to its left are in one of the states  $\text{green}$ ,  $\text{white}$ , or  $\bullet$ . If the test succeeds it crosses to  $\bullet$ ; otherwise it goes back to the initial state  $\text{green}$ . From  $\bullet$  it can perform a local transition and move to  $\text{yellow}$ . At  $\text{yellow}$ , the process looks left again and performs the same test as before. If successful, it will now move to  $\text{blue}$ . At  $\text{blue}$ , the process now looks right, and checks whether all processes to its right are in one of the states  $\text{green}$ ,  $\text{white}$ , or  $\bullet$ . If the test succeeds it crosses to its critical section  $\text{red}$ , from which it can go back to the initial state  $\text{green}$ .

The parameterized version of Burns' protocol (Fig. 2) consists of an arbitrary number of processes. The goal is to show that if we start from a configuration where all the processes are in state  $\text{green}$  then it is not possible to reach a configuration where two or more processes are in state  $\text{red}$ .

## 2.2 Transition System

We define the transition system induced by the parameterized version of Burns' protocol. More precisely, we define the set of *configurations* and the *transition relation*.

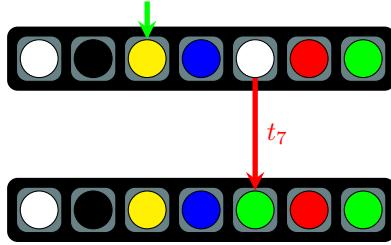
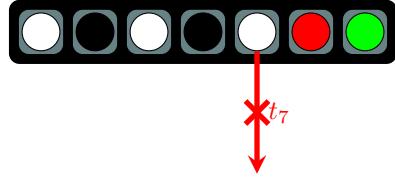
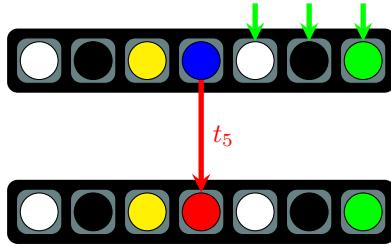
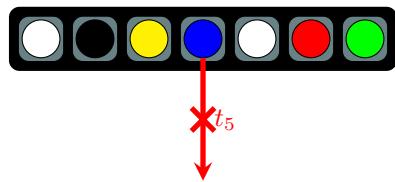
A configuration gives the states of the processes in a given instance of the system. A configuration in Burns' protocol is depicted in Fig. 3, corresponding to an instance of the system with four processes that are in states  $\text{white}$ ,  $\bullet$ ,  $\text{red}$ , and  $\text{blue}$  respectively. Notice that the set of configurations is infinite since there is no bound of the number of processes.

The transition relation is induced on the set of configurations by the above mentioned three types of transitions. When performing a transition, a process, called the *active process* changes state while the other processes remain passive (although their states may help enable/disable the move of the active process).

Fig. 4 depicts a local transition, where the active process performs  $t_3$  and changes state from  $\bullet$  to  $\text{yellow}$  while the states of the other processes remain unchanged.

An existential global transition is shown in Fig. 5. Here, the active process performs  $t_7$  and changes state from  $\text{white}$  to  $\text{green}$ . The transition is enabled since there is a witness in state  $\text{yellow}$  (marked by a green arrow). On the other hand, in Fig. 6, the transition is not enabled since there is no witness with the appropriate state to the left of the active process.

A universal global transition is shown in Fig. 7. The active process performs  $t_5$  and changes state from  $\text{blue}$  to  $\text{red}$ . The transition is enabled since all processes to the right of the

**Figure 5** An existential transition.**Figure 6** A disabled existential transition.**Figure 7** A universal transition.**Figure 8** A disabled universal transition.

active process (marked by green arrows) are in one of the states  $\bullet$ ,  $\circlearrowleft$ , or  $\bullet$ , as required by the condition of the transition. In Fig. 8, the same transition is not enabled since there is one process in state  $\bullet$  to the right of the active process (thus violating the condition of the transition).

For a configuration  $c$ , we use  $\text{post}(c)$  to be the set of configurations that we can reach from  $c$  through the application of a single transition.

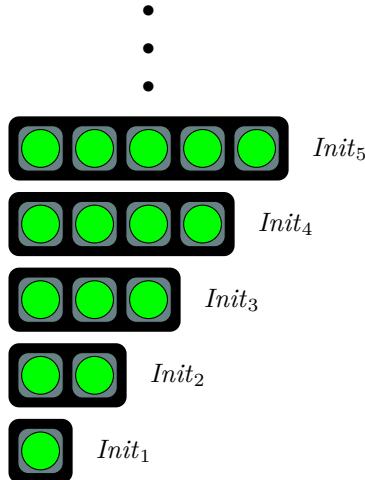
### 2.3 Safety Properties

Checking a safety property amounts to checking whether an instance of the system, starting from an *initial configuration*, can reach a *bad configuration* through repeated applications of the post operator.

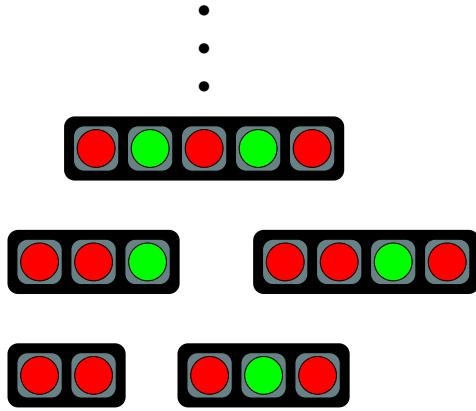
In Burns' protocol, an initial configuration (Fig. 9) contains only processes in the state  $\bullet$ . The set  $\text{Init}$  of initial configurations is infinite since there is one initial configuration for each size of the system. The set  $\text{Init}$  can be characterized by a (very simple) regular expression, namely  $(\bullet)^+$ , i.e., one or more processes in state  $\bullet$ .

A bad configuration (Fig. 10) is one which contains two or more processes in their critical sections (in state  $\bullet$ ). This set  $\text{Bad}$  of bad configurations is also infinite. The set  $\text{Bad}$  is upward closed wrt. the subword relation. Here, we say that a configuration  $c_1$  is subword of a configuration  $c_2$  if  $c_1$  occurs (not necessarily contiguously) in  $c_2$ . Obviously if a configuration contains at least two processes in state  $\bullet$  then any larger configuration (wrt. the subword relation) will also contain at least two processes in state  $\bullet$ , and hence belongs to the set of bad configurations. In fact, the set  $\text{Bad}$  is often characterized by its (finite) set  $\text{Bad}_{\min}$  of minimal elements. In the case of Burns' protocol,  $\text{Bad}_{\min}$  is the singleton containing the configuration  $\bullet\bullet$ .

Mutual exclusion is a safety property. Checking it amounts to checking whether there is a sequence of transitions that leads from an initial configuration to a bad configuration.



■ **Figure 9** The set of initial configurations.



■ **Figure 10** Examples of bad configurations.

### 3 Verification

We describe a scheme that allows to carry out parameterized verification automatically. The scheme consists of two procedures that can be performed in parallel, independently of each other. The first procedure is an under-approximation of the set of reachable configurations that is performed in the concrete domain. The second procedure is an over-approximation that is based on *view abstraction*. Both procedures are parameterized by a natural number  $k \geq 1$  that defines the degree of the precision of the approximation.

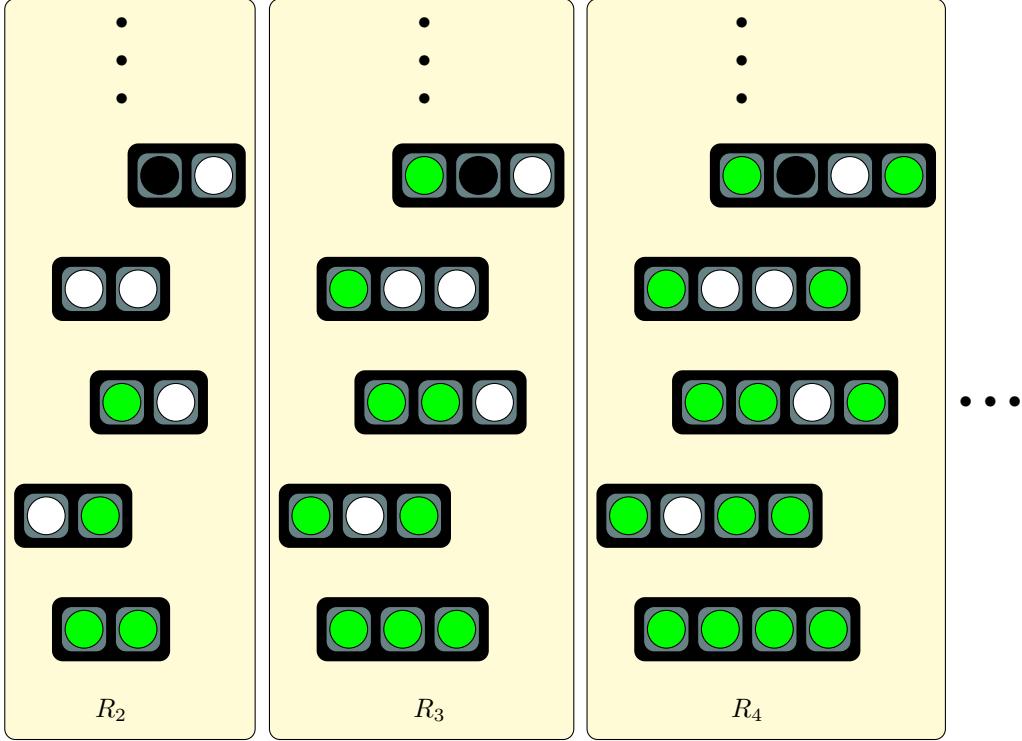
#### 3.1 Under-approximation

For a given  $k \in \mathbb{N}$ , we perform reachability analysis on the set of configurations of size  $k$  (Fig. 11). We start from the initial configuration of size  $k$  and generate all reachable configurations  $R_k$  of size  $k$ . This amounts to performing standard reachability analysis on a finite-state system since there are only finitely many configurations of size  $k$ . We can inspect  $R_k$  and search for bad configurations. We start from  $k = 1$ , and increase the value of  $k$  successively. If there is a bad configuration (of some size  $k$ ) that is reachable, then it will be detected by the under-approximation procedure when inspecting  $R_k$ . Consequently, if the system does not satisfy the safety property then this will be reported by the under-approximation procedure. However, the procedure is not able to prove correctness of the system, since this would require computing  $R_k$  for all  $k \in \mathbb{N}$ .

#### 3.2 Over-approximation

The over-approximation procedure is based on an abstract interpretation scheme, called *view abstraction*, that is parameterized by a natural number  $k \in \mathbb{N}$ . The concrete domain consists of the configurations of the system, while the abstract domain consists of objects that we call *views*. As we shall see below, each view is a subword of a configuration. We define an abstraction function  $\alpha_k$ , a concretization function  $\gamma_k$ , and an abstract post operator  $\text{Apost}_k$ , all of which are parameterized by  $k$ .

For a configuration  $c$ , the abstraction  $\alpha_k(c)$  is the set of views (subwords of  $c$ ) of size up to  $k$  (Fig. 12). For a set  $C$  of configurations, we define  $\alpha_k(C) := \cup_{c \in C} \alpha_k(c)$ . Consider a set

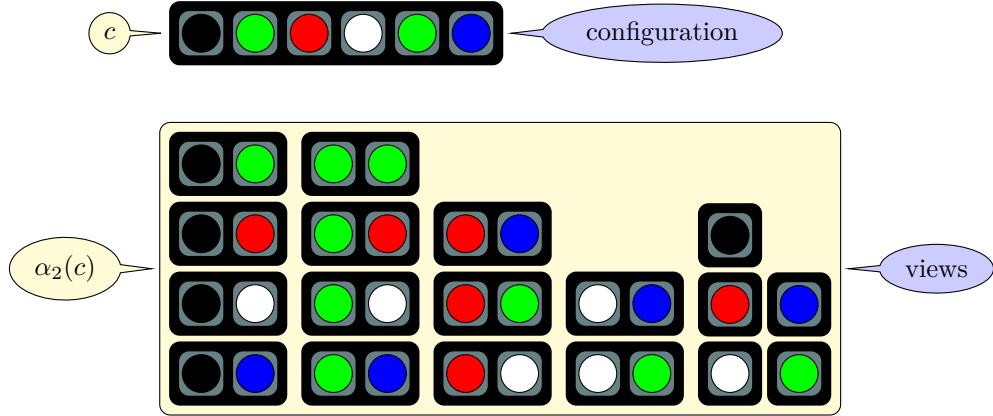


**Figure 11** The under-approximation procedure.

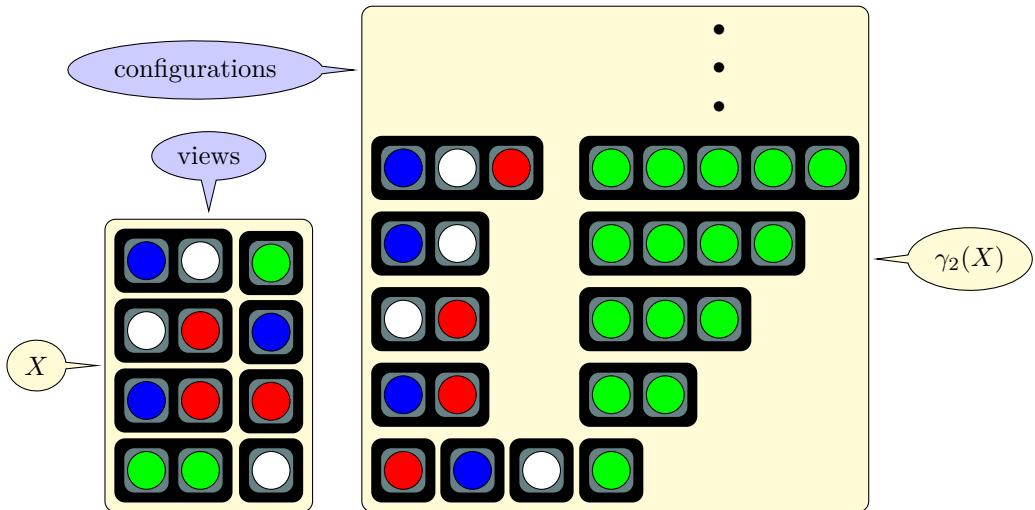
of views of size up to  $k$  such that  $X$  is downward closed, i.e., if  $X$  contains a view  $x$  then each subword of  $x$  is also included in  $X$ . We define the *concretization*  $\gamma_k(X)$  of  $X$  to be the set of configurations  $c$  such that  $\alpha_k(c) \subseteq X$ , i.e. all members of the abstraction of  $c$  are included in  $X$  (see Fig. 13.) Notice that, even for a finite set  $X$  and for a fixed  $k \in \mathbb{N}$ , the set  $\gamma_k(X)$  is in general infinite (as is the case with  $\gamma_2(X)$  in Fig. 13).

For  $k \in \mathbb{N}$ , we define the abstract post operator  $\text{Apost}$  such that for a set of views of size up to  $k$ , we have  $\text{Apost}_k(X) := \alpha_k(\text{post}(\gamma_k(X)))$ . In other words, we first take the concretization of  $X$ , then apply the concrete post operator, and finally take the abstraction of the result. We will perform reachability analysis on the set of views using a fixpoint iteration, parameterized with  $k \in \mathbb{N}$ . More precisely, we define the set  $V_k := \mu X. \alpha_k(\text{Init}) \cup \text{Apost}_k(X)$ . Before we describe how we compute  $V_k$ , we will give two of its properties (illustrated in Fig. 14). Let  $R$  be the set of reachable configurations. First, the concretization of  $V_k$  is an over-approximation of  $R$ , i.e.,  $R \subseteq \gamma_k(V_k)$  for all  $k \in \mathbb{N}$ . Furthermore, the precision of the abstraction increases with  $k$  in the sense that  $\gamma_{k+1}(V_{k+1}) \subseteq \gamma_k(V_k)$  for all  $k \in \mathbb{N}$ . We use these two properties to define the following scheme for proving correctness of the system. Suppose that the system is correct, i.e.,  $R \cap \text{Bad} = \emptyset$ . We consider the sequence of sets  $\gamma_1(V_1) \supseteq \gamma_2(V_2) \supseteq \gamma_3(V_3) \supseteq \dots$ . We start with  $\gamma_1(V_1)$  and check whether  $\gamma_1(V_1) \cap \text{Bad} = \emptyset$ . If the answer is positive then we know that the system is correct (since  $R \subseteq \gamma_k(V_k)$  and hence  $R \cap \text{Bad} = \emptyset$ .) On the other hand, if  $\gamma_1(V_1) \cap \text{Bad} \neq \emptyset$  (which is the case in Fig. 14) then we are not sure whether  $R \cap \text{Bad} \neq \emptyset$  holds or not. Therefore, we increase  $k$  and repeat the procedure for  $\gamma_2(V_2)$ . In Fig. 14, the intersection  $\gamma_2(V_2) \cap \text{Bad}$  is still not empty. Therefore, we increase  $k$  yet again. The procedure will terminate if and when we reach a  $k$  where  $\gamma_k(V_k) \cap \text{Bad} = \emptyset$ . In Fig. 14 such a  $k$  exists and  $k = 3$ .

Computing  $\text{Apost}_k(X)$  efficiently is not straightforward. The reason is that the set  $\gamma_k(X)$  is in general infinite even if the set of views  $X$  is finite. We solve this problem by showing

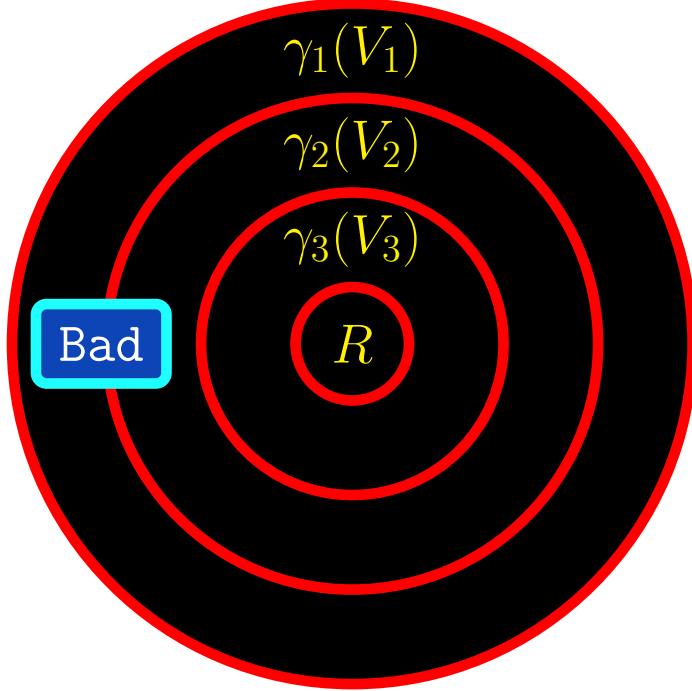


■ **Figure 12** A configuration and its  $\alpha_2$ -abstraction.



■ **Figure 13** Concretization of a set of views.

that we only need to perform a simpler operation  $\text{Apost}_k^{k+1}(X)$ . For  $1 \leq \ell \leq k$ , we define  $\text{Apost}_k^\ell(X) := \alpha_k(\text{post}(\gamma_k^\ell(X)))$ , where  $\gamma_k^\ell(X)$  is the subset of  $\gamma_k(X)$  containing only views of size up to  $\ell$ . Notice that for any  $\ell$  (and in particular for  $\ell = k+1$ ), the set  $\gamma_k^\ell(X)$  is finite and (easily) computable. We will illustrate why  $\text{Apost}_k(X) = \text{Apost}_k^{k+1}(X)$  through the following example which shows a part of computing  $\text{Apost}_2(X)$ . Assume that  $X$  contains (among others) the views and . We first compute the set  $\gamma_2^3(X)$ , i.e., we include all members of the concretization of  $X$  of size up to 3. For instance, the configuration is a member of  $\gamma_2^3(X)$ . Then, we apply the concrete post operator  $\text{post}$  on the set  $\gamma_2^3(X)$ . In particular we apply  $\text{post}$  on the configuration . For instance, if the active process is then transition  $t_8$  is enabled due to the existence of the witness to the left of the active process. As a result, we obtain the configuration . Finally, we apply the abstraction  $\alpha_2$ , obtaining, among others, the new views and . In particular, the latter view was obtained from the view which



■ **Figure 14** Abstract Analysis.

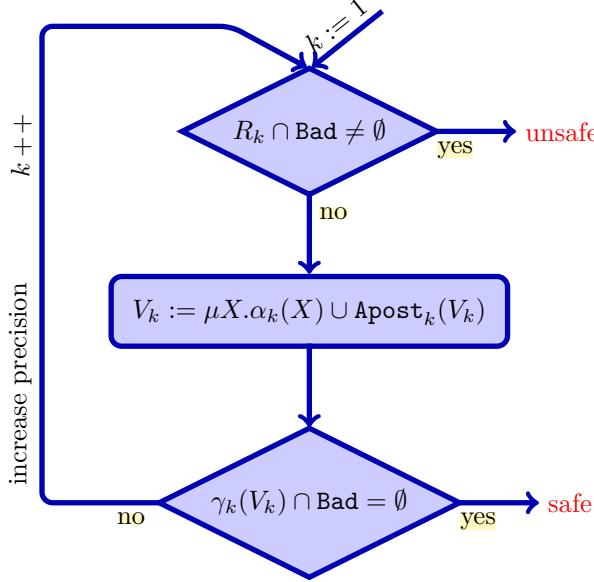
was part of  $X$ , through applying the transition  $t_8$ . There are two interesting aspects of this transition to observe. First, the transition needed the witness which was not part of the original view. Second, the witness consists of a *single* process. Thus, in order to derive the new view we needed to add *one* extra process in order to accommodate the witness. This explains the reason why we need to consider concrete configurations of larger sizes than the views (to ensure the inclusion the witnesses), but also the reason why we need to only consider configurations whose sizes are larger by *one* (since witnesses are of size one).

### 3.3 Scheme

Our verification scheme consists of performing a number of iterations, where each iteration corresponds to a particular value of  $k \in \mathbb{N}$  (Fig. 15). We start with  $k = 1$ . During each iteration, we run the under- and over-approximation procedures for the given value of  $k$ . If the under-approximation procedure is conclusive, i.e.,  $R_k \cap \text{Bad} \neq \emptyset$  then we terminate and declare the system *unsafe*. Otherwise, we run the over-approximation procedure to compute  $V_k$ . If this is conclusive, i.e.,  $V_k \cap \text{Bad} = \emptyset$  then we terminate and declare the system *safe*. Notice that if either of the two procedures is conclusive then the current  $k$  is a *cut-off* point beyond which we need not continue (since we have either concluded correctness or incorrectness of the system). If none of the procedures is conclusive, we increase the precision, by increasing the value of  $k$ , and perform the next iteration.

We show how to perform each operation that is required for implement the scheme.

- *Checking whether  $R_k \cap \text{Bad} \neq \emptyset$ .* As mentioned above, computing  $R_k$  amount to performing reachability analysis on a finite-state system. Furthermore, to check  $R_k \cap \text{Bad} \neq \emptyset$  we need only to consider elements of  $\text{Bad}$  whose sizes are up to  $k$ . The latter is finite and easily computable since  $\text{Bad}$  is upward closed. Thus, we need to consider the intersection of two finite sets.



**Figure 15** Abstract Analysis.

- *Computing  $V_k$ .* We can compute the fixpoint as follows.
  - Since the set of initial configurations  $\text{Init}$  is regular, computing  $\alpha_k(\text{Init})$ , for any  $k \in \mathbb{N}$ , amounts to generating the subwords of members of  $\text{Init}$  of size up to  $k$ . This is a task that can be accomplished using simple automata operations.
  - As mentioned above, for a given set of views  $X$ , we need only to compute  $\text{Apost}_k^{k+1}(X)$  rather than  $\text{Apost}_k(X)$ . The former can be computed as follows: (i) we compute the finite set of configurations  $C := \gamma_k^{k+1}(X)$  by matching the different members of  $X$ . (ii) We compute the set  $C' := \text{post}(C)$  by applying the transition relation on  $C$ . Notice that  $C'$  is finite. (iii) We compute the finite set of views  $V' := \alpha_k(C')$  which amounts to computing the abstraction of a finite set.
- *Checking whether  $V_k \cap \text{Bad} = \emptyset$ .* This amounts to checking whether there is a minimal configuration  $c$  in  $\text{Bad}$  (i.e.,  $c \in \text{Bad}_{\min}$ ) such that  $\alpha_k(c) \subseteq V_k$ . Recall that  $\text{Bad}_{\min}$  is finite and given, and hence we can perform the test by systematically going through all members of  $\text{Bad}_{\min}$ .

## 4 Application

We illustrate the verification scheme by applying it to Burns' protocol.

### Iteration 1: Under-Approximation.

We start from the initial configuration of size one. The set  $R_1$  of reachable configurations of size one can be shown to contain all configurations of size one. However, none of these configurations belongs to  $\text{Bad}$  since each member of  $\text{Bad}$  contains at least two processes (in state ). Hence, the under-approximation scheme is inconclusive for  $k = 1$ .

### Iteration 1: Over-approximation.

The set  $\alpha_1(\text{Init})$  contains a single view, namely . This will be added to  $V_1$ . In order to compute the fixpoint, we apply  $\text{Apost}_1^2$  repeatedly. Let us consider its first application.

The set  $\gamma_1^2$  contains the two configurations and . Applying  $\text{post}$  to these two configurations gives the configurations , , and . Applying  $\alpha_2$  on the derived set of configurations gives the set containing the two views and that will now be added to  $V_1$ . Applying  $\text{Apost}_1^2$  repeatedly in this manner will yield the set of all views of size one. Applying  $\gamma_2$  to this set gives the set of *all* configurations, and in particular this set will intersect with the set  $\text{Bad}$ . Therefore, the over-approximation scheme is inconclusive for  $k = 1$ .

### Iteration 2: Under-Approximation.

We start from the initial configuration of size two. The set  $R_2$  of reachable configurations of size two can be computed using finite-state reachability analysis. We leave the computation of the set (as an easy exercise) to the reader. None of the configurations in  $R_2$  belongs to  $\text{Bad}$ , and hence, the under-approximation scheme is inconclusive also for  $k = 2$ .

### Iteration 2: Over-approximation.

The set  $\alpha_2(\text{Init})$  contains the two views and . Thus, these views will be added to  $V_2$ . In order to compute the fixpoint, we apply  $\text{Apost}_2^3$  repeatedly. Let us consider its first application. The set  $\gamma_2^3$  contains the configurations , , and . Applying  $\text{post}$  to these three configurations gives the configurations , , , , , and . Applying  $\alpha_2$  to the derived set of configurations gives the set containing the views , , , , , and . Applying  $\text{Apost}_2^3$  repeatedly will yield the set of all views of size two *except* the views and . In particular, the absence of the second view implies that applying  $\gamma_2$  gives a set of configurations that does not intersect with the set  $\text{Bad}$ . This means that we can terminate and conclude that the system is safe. Notice that the cut-off point for Burns' protocol is  $k = 2$ .

## 5 Completeness

We will give examples of systems for which our method is complete (for which the scheme is guaranteed to terminate). First, we give the definitions of upward and downward closed sets, and then give a sufficient condition that guarantees termination. Finally, we describe a class of systems, namely *monotonic systems*, that satisfy the condition.

### 5.1 Downward and Upward Closed Sets

Let  $\preceq$  be the subword relation on configurations. The relation  $\preceq$  is a *well quasi-ordering* [23]: for any infinite sequence  $c_0, c_1, c_2, \dots$  of configurations, there are  $i < j$  such that  $c_i \preceq c_j$ . A set  $D$  of configurations is said to be *downward closed* if  $c_1 \in D$  and  $c_2 \preceq c_1$  implies  $c_2 \in D$ . A

set  $U$  of configurations is said to be *upward closed* if  $c_1 \in U$  and  $c_1 \preceq c_2$  implies  $c_2 \in U$ . For a set  $C$  of configurations, we define  $C\uparrow := \{c_2 \mid \exists c_1 \in C. c_1 \preceq c_2\}$  to be the upward closure of  $C$ . Let  $\min(C)$  be the set of minimal elements of  $C$ . The set  $\min(C)$  is always finite. Notice that, for an upward closed set  $U$ , we have that  $U\uparrow = U$ , and that  $U$  can be characterized by its minimal elements in the sense that  $U = \min(U)\uparrow$ . For instance, the set **Bad** in Burns' protocol is upward closed, and  $\min(\text{Bad})$  is a singleton containing the configuration . Observe that the complement  $\neg D$  of a downward closed set  $D$  is upward closed, and vice versa.

## 5.2 Sufficient Condition

Let  $D$  be a set of configuration. We say that  $D$  is a *good downward closed invariant* if it satisfies the following conditions: (i) *downward closed*:  $D$  is downward closed; and (ii) *invariant*:  $D$  is inductive, i.e.,  $D$  contains the set **Init** of initial configuration, and  $D$  is closed under the application of the transition relation (for any  $c \in D$ , a transition from  $c$  leads to a configuration inside  $D$  again.) (iii) *good*:  $D \cap \text{Bad} = \emptyset$ . If  $D$  satisfies conditions (i) and (ii) then we can show that there is a  $k$  such that  $\gamma_k(V_k) \subseteq D$ . In fact, we can define  $k$  to be the size of a largest configuration in  $\min(\neg D)$ , i.e., the size of a largest configuration in the minimal set of configurations in the complement of  $D$ . Recall that  $\neg D$  is upward closed. If  $D$  also satisfies condition (iii) then the over-approximation procedure is guaranteed to terminate. More precisely,  $D \cap \text{Bad} = \emptyset$  implies that  $\gamma_k(V_k) \cap \text{Bad} = \emptyset$ , which means that the procedure declares correctness of the system at step  $k$  (if not earlier). As a side remark, notice also that if the set  $R$  of reachable configurations is downward closed then  $\gamma_k(V_k) = R$  for some  $k$  (we know from the previous section that  $R \subseteq \gamma_k(V_k)$  for all  $k$ .)

We will motivate that if  $R$  is downward closed then our procedure is guaranteed to terminate implying its completeness. We consider two cases. If  $R \cap \text{Bad} \neq \emptyset$ , then there is a  $k$  such that  $R_k \cap \text{Bad} \neq \emptyset$  and the under-approximation procedure declares the system unsafe during the  $k^{th}$  iteration. If  $R \cap \text{Bad} = \emptyset$  then since  $R$  is an invariant, i.e., it satisfies the conditions (i), (ii), and (iii), and hence the over-approximation procedure will terminate.

In the case of Burns' protocol, the set  $R$  is characterized by the regular expression  $(\text{●} + \text{○} + \text{●} + \text{○} + \text{●})^* \cdot (\text{●} + \epsilon) \cdot (\text{●} + \text{○} + \text{●} + \text{○})^*$ . Our method will not compute this regular expression explicitly. However, the language of the expression is downward-closed and is equal to the set  $\neg \left\{ \text{●●}, \text{●●○} \right\}\uparrow$ . The size of the largest elements in the set of minimal configurations is equal to 2, which explains why the over-approximation procedure terminates at  $k = 2$  for Burns' protocol.

## 5.3 Monotonic Systems

A system is said to be *monotonic* if, for all configurations  $c_1, c_2, c_3$ , whenever  $c_2 \in \text{post}(c_1)$  and  $c_1 \preceq c_3$  then there is a configurations  $c_4$  such that  $c_2 \preceq c_4$  and  $c_4 \in \text{post}(c_3)$ . In other words, the relation  $\preceq$  is a simulation wrt. the transition relation. For monotonic systems, it is the case that  $R \cap \text{Bad} = \emptyset$  iff  $R\downarrow \cap \text{Bad} = \emptyset$ . This follows from the assumption that **Bad** is an upward closed set. Therefore, taking the downward closure of the set of reachable configurations does not cause any imprecision. This means that we can work with a new transition relation in which we allow the system to be *lossy*: a configuration may, at any point of time, lose an arbitrary number of processes. The new transition relation will reach **Bad** if and only if the old one does. Furthermore, the new set of reachable configurations is

downward closed which means that our procedure is guaranteed to terminate. In fact, there is a wide class of systems that induce transition relations that are monotonic with respect to a well quasi-ordering. Our scheme is complete for such systems. Examples include lossy channel systems [8], Petri nets (our procedure solves the coverability problem for them), timed Petri nets [10], etc.

## 6 Conclusion

We have presented a method for automatic verification of parameterized systems. The method proves or refutes whether a given safety property is satisfied by only inspecting small instances of the system. More precisely, we run two procedures in parallel. The first computes the (finite) set of reachable configurations of size up to  $k$  for some natural number  $k$ . The second procedure carries out an abstract interpretation scheme, called *view abstraction*, in which precision is defined (and can be increased) using a natural number  $k$ . The latter is guaranteed to terminate in case there is a good invariant that is downward closed. This implies that the whole method is guaranteed to terminate in case the set of reachable configurations is downward closed and hence that termination is guaranteed in case the transition relation is monotonic on the set of configurations.

The framework can be extended in a straightforward manner to other types of topologies such as multisets, rings, and trees, and also extended to the case where global transitions are not assumed to happen atomically [4].

---

### References

---

- 1 P. A. Abdulla and G. Delzanno. On the coverability problem for constrained multiset rewriting. In *Proc. AVIS'06, 5<sup>th</sup> Int. Workshop on Automated Verification of Infinite-State Systems*, 2006.
- 2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Carl Leonardsson, and Ahmed Rezine. Counter-example guided fence insertion under tso. In *Proc. TACAS '08, 18<sup>th</sup> Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, 2012.
- 3 Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Proc. LICS '04, 20<sup>th</sup> IEEE Int. Symp. on Logic in Computer Science*, pages 345–354, 2004.
- 4 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. All for the price of few (parameterized verification through view abstraction). In *VMCAI*, volume 7737 of *LNCS*, pages 476–495, 2013.
- 5 Parosh Aziz Abdulla, Frédéric Haziza, Lukás Holík, Bengt Jonsson, and Ahmed Rezine. An integrated specification and verification technique for highly concurrent data structures. In *TACAS*, volume 7795 of *LNCS*, pages 324–338, 2013.
- 6 Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, and Ahmed Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *Proc. TACAS '07, 13<sup>th</sup> Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 721–736. Springer Verlag, 2007.
- 7 Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, and Ahmed Rezine. Handling parameterized systems with non-atomic global conditions. In *Proc. VMCAI '08, 9<sup>th</sup> Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, 2008.
- 8 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *LICS*, pages 160–170. IEEE Computer Society, 1993.

- 9 Parosh Aziz Abdulla, Axel Legay, Julien d'Orso, and Ahmed Rezine. Simulation-based iteration of tree transducers. In *Proc. TACAS '05, 11<sup>th</sup> Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, 2005.
- 10 Parosh Aziz Abdulla and Aletta Nyblén. Timed Petri nets and BQOs. In *Proc. IC-ATPN'2001: 22nd Int. Conf. on application and theory of Petri nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70, 2001.
- 11 T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In Berry, Comon, and Finkel, editors, *Proc. 13<sup>th</sup> Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 221–234, 2001.
- 12 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18. ACM, 2010.
- 13 Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proc. 15<sup>th</sup> Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.
- 14 A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. 16<sup>th</sup> Int. Conf. on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386, Boston, July 2004. Springer Verlag.
- 15 D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, 2001.
- 16 G. Delzanno. Automatic verification of cache coherence protocols. In Emerson and Sistla, editors, *Proc. 12<sup>th</sup> Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag, 2000.
- 17 Giorgio Delzanno. Verification of consistency protocols via infinite-state symbolic model checking. In *FORTE'00*, volume 183 of *IFIP Conference Proceedings*, pages 171–186. Kluwer, 2000.
- 18 J. Esparza and S. Schwoon. A bdd-based model checker for recursive programs. In *CAV*, volume 2102 of *LNCS*, pages 324–336. Springer, 2001.
- 19 Pierre Ganty, Jean-François Raskin, and Laurent Van Begin. A Complete Abstract Interpretation Framework for Coverability Properties of WSTS. In *VMCAI'06*, volume 3855 of *LNCS*, pages 49–64. Springer, 2006.
- 20 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge and check... made efficient. In *CAV'05*, volume 3576 of *LNCS*, pages 394–407. Springer, 2005.
- 21 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of wsts. *J. Comput. Syst. Sci.*, 72(1):180–203, 2006.
- 22 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.
- 23 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.* (3), 2(7):326–336, 1952.
- 24 Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.
- 25 Kedar S. Namjoshi. Symmetry and completeness in the analysis of parameterized systems. In *VMCAI'07*, volume 4349 of *LNCS*, pages 299–313. Springer, 2007.
- 26 A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Proc. TACAS '01, 7<sup>th</sup> Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031, pages 82–97, 2001.

- 27 A. Pnueli, J. Xu, and L. Zuck. Liveness with (0,1,infinity)-counter abstraction. In *Proc. 14<sup>th</sup> Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.
- 28 Susmit Sarkar, Peter Sewell, Jade Alglave, Luc Maranget, and Derek Williams. Understanding power multiprocessors. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 175–186, 2011.
- 29 T. Touili. Regular Model Checking using Widening Techniques. *Electronic Notes in Theoretical Computer Science*, 50(4), 2001. Proc. of VEPAS'01.
- 30 N. Yonesaki and T. Katayama. Functional specification of synchronized processes based on modal logic. In *IEEE 6th International Conference on Software Engineering*, pages 208–217, 1982.