Model Checking Large-Scale and Parameterized Resource Allocation Systems

E. Allen Emerson and Vineet Kahlon

Department of Computer Sciences*
The University of Texas at Austin, Austin TX-78712, USA

Abstract. In this paper, techniques are proposed for limiting state explosion in the context of resource allocation problems. It is shown that given any system organized into a — possibly irregular — network of n — possibly heterogeneous — processes, model checking over that system can be reduced by an efficient, fully automatic and exact method to model checking over a certain small system. These results are established for correctness properties expressed in LTL$\backslash$X. The precise size and topology of the small system are dependent on the large system, as well as the correctness specification. When the network is symmetric and the processes homogeneous, this new method provides an efficient solution to the Parameterized Model Checking Problem. As an application, it is shown how to efficiently verify a variety of solutions to the parameterized Dining Philosophers Problem.

1 Introduction

Model checking has become a widely used method of verification. A key limitation to its use is the state explosion problem. A variety of techniques to limit state explosion have been investigated. Among these are techniques such as symmetry reduction (cf. e.g., [7], [16]) which typically focuses on large but constant size systems of homogeneous processes configured in a regular or symmetric interconnection topology. Other techniques include partial order methods ([20], [24]) and Binary Decision Diagrams ([4],[5]), which attempt to exploit regular organization embedded in the designs of systems.

In this paper, we first propose new methods for limiting state explosion when reasoning about potentially irregular systems, i.e., systems with an irregular network topology comprised of many heterogeneous processes. We go on to show how our methods can be specialized to reason about regular systems: systems comprised of many homogeneous processes arranged in a regular network. A benefit is a new approach to reasoning about parameterized systems, as described below.

We work in the context of resource allocation problems which are designed for the purpose of resolving conflicts in concurrent systems, a problem of considerable practical importance. We consider resource allocation systems organized into a (possibly irregular) network of (possibly heterogeneous) processes. The processes interact

* This work was supported in part by NSF grant CCR-009-8141, SRC contract 99-TJ-685, and TARP-003685-0650-1999. The authors can be reached at {emerson,kahlon}@cs.utexas.edu

by sharing resources, which are represented by *tokens*. Each edge connecting a pair of processes P_i and P_j is labeled by a token t which is shared between the two processes. Our results easily generalize to the case where a resource could be shared by more than two processes, but for the sake of simplicity, we restrict ourselves to the case where each resource is shared by exactly two processes. A process can perform certain independent actions internally, without possession of its tokens. But certain other actions require that the process acquire all of its neighboring tokens. Once it has them, it can perform token-guarded actions. Eventually a process releases all its tokens, returning them to their associated edges.

We show that model checking over any such large system can be reduced to model checking over an equivalent small system. The precise size and topology of the small system are dependent on the large system, as well as the number of indices (i.e., processes tracked) in the correctness specification. The reduction method is efficient, fully automatic and exact. We establish our results for correctness properties expressed in LTL$\backslash$X.

All of the above techniques deal with reducing the cost of model checking a single large system, e.g., a system with k processes for some large constant k. In contrast, we might want to verify that a uniform infinite family of such systems is correct for all sizes n. The special case of such parameterized systems regularly organized with arbitrary number of homogeneous processes in a symmetric network is an important one with many applications. And our results specialize to apply to the associated *Parameterized Model Checking Problem (PMCP)*, which is the following: to decide whether a temporal property is true for every size system in a uniform family of systems. A key advantage is that one application of the PMCP settles the scalability and the state explosion problem in one fell swoop. We illustrate our techniques by applying them to the verification of a variety of parameterized Dining Philosophers solutions. To our knowledge, this is the first such fully automated verification for the parameterized formulation of this problem. We are able to reduce reasoning about the correctness for Dining Philosophers protocols of all sizes n to reasoning about at most 5 philosophers.

The rest of this paper is organized as follows: The model of computation is given in section 2. In section 3, we show how to reduce model checking over large resource allocation systems to small ones for correctness specifications involving finite behavior. Results involving unconditionally fair behaviour are considered in section 4. Deadlock issues are considered in section 5, while example applications are given in section 6. Some concluding remarks are given in section 7.

2 Preliminaries

2.1 Model of Computation

We consider *systems* $\mathcal{U} = (\mathcal{P}, \mathcal{T})$, where $\mathcal{P}$ is a set of a finite number n of possibly heterogeneous *processes* $P_1, ..., P_n$ and $\mathcal{T}$ is a set of *tokens* $t_1, ..., t_m$ representing resources shared between pairs of processes. The system $\mathcal{U}$ may be viewed as an undirected graph or network with node set $\mathcal{P}$ and edge set $\mathcal{T}$. In any global state of $\mathcal{U}$, each token of $\mathcal{T}$ can be in the possession of at most one process of $\mathcal{U}$. A token not in the possession of any process is said to be *free* in that global state. The set of tokens adjacent to and used by P_i is denoted by $Tok(P_i)$. We assume that the processes in $\mathcal{P}$ execute concurrently with interleaving semantics, competing for shared token resources, permitting access to guarded regions of code. The syntax of each process P_i ensures that each of its executions is of the form:

((internal transition)* $acquire$ (internal transition)* $release$)$^+$.

An *internal* transition of P_i only changes its local state, and not the status of its adjacent tokens. The *external* transitions of P_i are the *acquire* and *release* transitions, which respectively grab free adjacent tokens and release them. We also classify transitions into two types: *token free* transitions of process P_i can always be executed irrespective of the current global state of the system; *token dependent* transitions of process P_i can can be executed only if P_i currently possesses all tokens in the set $Tok(P_i)$. Transitions bracketed within an *acquire-release* pair, including *release* but excluding *acquire* are *token depedent*. Note that a process P_i is not pre-emptible for tokens it is presently using. Once a token in $Tok(P_i)$ is obtained by an *acquire* transition of P_i, it cannot be released until all tokens in $Tok(P_i)$ have been obtained and the corresponding token *release* transition executed.

Each set $Tok(P_i)$ is partitioned into subsets $Tok(P_i, 1), ..., Tok(P_i, k(P_i))$. We assume that a partial order $\prec_{P_i}$, or $\prec_i$, for short, is imposed on these subsets. Process P_i when executing an *acquire* transition, accumulates its tokens in a manner consistent with this partial order and partitioning of $Tok(P_i)$, as described below.

A process P_i is described by its *synchronization skeleton* [8], formally given by the tuple $(S_i, \Sigma_i, R_i, \mathsf{i}_i)$, where S_i is the set of local states of P_i, R_i is the transition relation of P_i, i_i is the initial state of P_i and $\Sigma_i = \Sigma_i^D \cup \Sigma_i^F \cup \{release_i, acquire_i\}$, where Σ_i^D and Σ_i^F represent the labels of the *token dependent* and the *token free* transitions, respectively. Permitted transitions in R_i are:

- $l_i \xrightarrow{a} m_i$: an internal local transition from local state l_i to local state m_i of process P_i labeled by $a \in \Sigma_i^D \cup \Sigma_i^F$.
- $l_i \xrightarrow{acquire_i} m_i$: a token acquire transition. When process P_i is at location l_i, it can fire this "molecular" transition. In each atomic step, P_i can acquire all tokens in $Tok(P_i, j)$, for some $j \in [1..k(P_i)]$, in case all tokens in the subset $Tok(P_i, j)$ are free; acquisition of these token sets must respect the partial order $\prec_i$ so that if for $p, q \in [1 : k(P_i)]$, $Tok(P_i, p) \prec_i Tok(P_i, q)$, then the set $Tok(P_i, p)$ must be acquired before the set $Tok(P_i, q)$. Once P_i possesses all tokens in $Tok(P_i)$, it transits to local state m_i. For multiple token sets, if process P_i has acquired *some* but not all of its token sets in $Tok(P_i)$, then execution appears to stutter in local state l_i. In other words, once P_i has acquired a token in state l_i it waits in l_i until all tokens in $Tok(P_i)$ have been acquired; and
- $l_i \xrightarrow{release_i} m_i$: a token release transition, which frees all tokens in $Tok(P_i)$.

We now define the semantics of the system $\mathcal{U} = (\mathcal{P}, \mathcal{T})$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$ and $\mathcal{T} = \{t_1, \ldots, t_m\}$. The possession status of token t_l shared by the two processes P_i and P_j is an element of the status set $T_l = \{i, j, \mathsf{free}\}$ corresponding to the token being possessed by process P_i or process P_j or being possessed by no process at all. It is convenient to denote the status of token t_l in global state s, by $status(s, t_l)$. We denote the set of tokens possessed by process P_i in global state s by $Pos(P_i, s)$. For process P_i and set $T \subseteq Tok(P_i)$, we use the phrase "T not grabbed at s" to mean that there exists $t \in T$, such that $status(s,t) \neq i$ and the phrase "T grabbed by process P_i at s" to mean that for each token $t \in T$, we have that $status(s,t) = i$.

Formally, we define the labeled transition system $\mathcal{M}_{\mathcal{U}}$ for $\mathcal{U}$ to be the tuple $(S, \Sigma, R, \mathsf{i})$, where

- $S = S_1 \times ... \times S_n \times T_1 \times ... \times T_m$. A global state s is a tuple of the form $(s(1), ..., s(n), s(n+1), ..., s(n+m))$ where components $s(1), \ldots, s(n)$ indicate respectively the local states of processes $P_1, ..., P_n$ while components $s(n+1), \ldots, s(n+m)$ describe respectively, the possession status of tokens $t_1, ..., t_m$; note that $status(s, t_j) = s(n+j)$.
- $\Sigma = \bigcup_i \Sigma_i$.
- $R = \{s \xrightarrow{a} t \mid \exists i : \exists a \in \Sigma_i^D \cup \Sigma_i^F : \exists l_i \xrightarrow{a} m_i \in R_i :\ s(i) = l_i$ and $t(i) = m_i$ and $\forall j \neq i :\ s(j) = t(j)\}$
  $\cup \{s \xrightarrow{acquire_i} t \mid i \in [1:n]$ and $\exists l_i \xrightarrow{acquire_i} m_i \in R_i :\ s(i) = l_i$ and
  $Tok(P_i, j)$ not grabbed at s implies
  $\forall p :\ Tok(P_i, j) \prec_i Tok(P_i, p) : Tok(P_i, p)$ not grabbed at s
  and $Tok(P_i)$ grabbed at s implies $t(i) = m_i$
  and $Tok(P_i)$ not grabbed at s implies $t(i) = l_i\}$
  $\cup \{s \xrightarrow{release_i} t \mid i \in [1:n]$ and $\exists l_i \xrightarrow{release_i} m_i \in R_i :\ s(i) = l_i$ and
  $t(i) = m_i$ and $\forall u \in Tok(P_i) : status(s,u) = i$,
  $status(t,u) = \mathsf{free}$ and $\forall v \notin Tok(P_i) : status(s,v) = status(t,v)\}$
- $\mathsf{i} = (\mathsf{i}_1, ..., \mathsf{i}_{n+m})$, where for all $j \in [1..m]$, $\mathsf{i}_{n+j} = \mathsf{free}$.

Process P' of system $\mathcal{V}$ is called a *replica* of process P of another system $\mathcal{U}$ iff the LTS corresponding to P is the same as the LTS corresponding to P'; however, the context, i.e., tokens sets $Tok(P)$ and $Tok(P')$ may be different.

2.2 Computation Paths and Stuttering

A *path* x of $\mathcal{U}$ is a sequence $s_0, s_1, \ldots$ of global states of $\mathcal{U}$ such that for every i, (s_i, s_{i+1}) corresponds to a transition of a process of $\mathcal{U}$. A *fullpath* x of $\mathcal{U}$ is a maximal path, viz., either it is infinite or the last state s_k, say, has no successor in $\mathcal{U}$ as no process is enabled in s_k. In this case the system is said to be *globally deadlocked* in s_k. A *computation* is a path that starts at the initial state i of $\mathcal{U}$. We say that the sequence of global states $y = t_0, t_1, \ldots$ is a *stuttering* of a path x iff y is of the form $z_0 z_1 ...$ such that for each j there is an $r > 0$ with $z_j = (x_j)^r$ (cf. [3]). Sequences u and v of global states of $\mathcal{U}$ are said to be *stuttering equivalent* iff there exists a path w of $\mathcal{U}$ such that each of u and v is a stuttering of w.

2.3 Temporal Logic

We assume that our correctness specifications are formulated using LTL\X (Linear Temporal Logic minus Next-time). Formulae h are built up from the usual linear operators F (sometime), G (always), and U (weak until) combined with propositional connectives. If h is a LTL\X formula then, for purposes of model checking, we have a very flat temporal logic that can express both $\mathsf{A}h$ (for all futures h) and $\mathsf{E}h$ (for some future h). As is usual for linear time, there is an implicit universal quantification over all possible futures. We write $\mathcal{U} \models h$ to mean that $\mathcal{U} \models \mathsf{A}h$ which is defined by $\mathcal{M}_{\mathcal{U}}, \mathsf{i} \models \mathsf{A}h$. Dually, $\mathcal{U} \models \mathsf{E}h$ is defined as $\mathcal{M}_{\mathcal{U}}, \mathsf{i} \models \mathsf{E}h$. The atomic propositions allowed in h are typically indexed; e.g., l_i means that process i is in local state l.

3 Reduction Results for Finite Behavior

In this section, we consider behavior along finite paths, with correctness formulated using a formula h of the logic LTL$\backslash$X. We use $\mathsf{E_{fin}}$ to denote existential quantification over finite paths, and $\mathsf{A_{fin}}$ to denote universal quantification over finite paths. Specifically, we focus on double index properties f of the form $\mathsf{E_{fin}}h(i,j)$ or its dual $\mathsf{A_{fin}}h(i,j)$, where $h(i,j)$ is a LTL$\backslash$X formula with atomic propositions over the local states of processes P_i and P_j. Our results will permit us to reason about safety properties.

Theorem 3.1 *Let $\mathcal{U}$ be a system with processes P_1, P_2, and $\mathcal{V}$ be a system with respective replica processes P_1', P_2' such that $Tok(P_1) \cap Tok(P_2) = \emptyset$ in $\mathcal{U}$ iff $Tok(P_1') \cap Tok(P_2') = \emptyset$ in $\mathcal{V}$. Then $\mathcal{U} \models \mathsf{E_{fin}}h(1,2)$ iff $\mathcal{V} \models \mathsf{E_{fin}}h(1,2)$.*

Proof Sketch ($\Rightarrow$) Given a finite path x of $\mathcal{U}$, we will construct a finite path y of $\mathcal{V}$ such that x projected onto coordinates 1 and 2, viz., processes P_1 and P_2, is stuttering equivalent to y projected onto 1 and 2. We present an informal description first. Along y all processes of $\mathcal{V}$ other than P_1' and P_2' idle in their initial states. To establish stuttering equivalence of x and y on 1 and 2, we ensure that P_1' and P_2' execute along y all the same visible local transitions in the same order as do respectively P_1 and P_2 along x, i.e. transitions causing a change in the local states along coordinates 1 and 2. With this in mind, we define an *event* of x to be a local transition along x that causes a change in the local states of process P_1 or P_2. We construct y in an inductive fashion by "scanning" for events of x starting from the initial state and mimicking them in the same order to get y.

In more detail, the construction is as follows. To start with y is initialized to $\mathsf{i}_{\mathcal{V}}$, the initial state of $\mathcal{V}$. All subsequent invisible non-event transitions along x are skipped over. For the inductive step, assume that the next event e encountered along x is a transition fired by, say, process P_1. A similar argument applies if it is fired by P_2. First, if event e is a token free or token dependent transition, then we let replica P_1' execute the same transition. Next, if the event e is a release transition then we let P_1' execute the corresponding release transition. The only subtlety occurs when event e corresponds to a token acquisition transition of P_1. That transition may fire several times along x; only the last such firing causes a visible change in the local state upon grabbing the last free block of tokens. (There may be a "race" between P_1 and P_2 to grab all their respective tokens, and the intervals during which they are trying may overlap. But along x, P_1 wins by getting all of its tokens first.) Hence, starting at the last global state of y constructed so far, we let P_1' execute a series of token acquisition transitions to acquire all the tokens in $Tok(P_1')$. (If, having lost the race, P_2 eventually acquires all its tokens

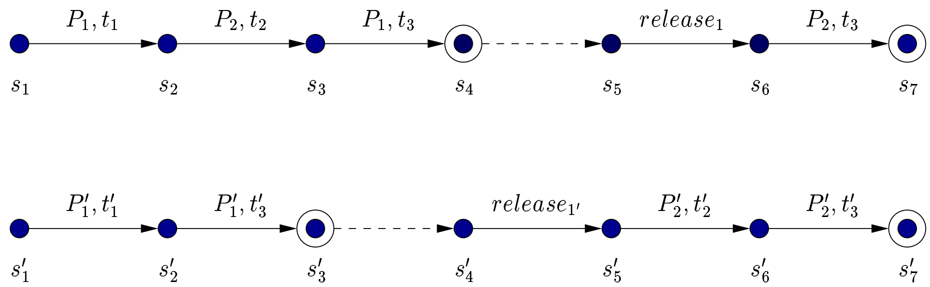

Fig. 1. Process P_1, with the token set $\{t_1, t_3\}$, completes acquisition of all of its tokens in state s_4 before P_2, which has the token set $\{t_2, t_3\}$, completes its in state s_7. We mimic this by letting P_1' acquire all of its tokens first and only when P_1' has released them in state s_5' do we schedule P_2' to acquire all of its tokens.

along x, then that defines a subsequent event, which will be mimicked then along y by P_2'.) This completes the construction of y. The idea is illustrated using figure 1.

($\Leftarrow$) Analogous to the above argument. $\square$

Given processes P_1 and P_2 of $\mathcal{U}$, we now define two small systems $\mathcal{V}_{\mathsf{c}}(P_1, P_2)$ and $\mathcal{V}_{\mathsf{nc}}(P_1, P_2)$ with P_1' and P_2', that are replicas of P_1 and P_2, respectively, sharing a common token t in $\mathcal{V}_{\mathsf{c}}(P_1, P_2)$ but sharing no token in $\mathcal{V}_{\mathsf{nc}}(P_1, P_2)$. Furthermore, in both systems we have, $\prec_{P_1'} = \emptyset = \prec_{P_2'}$. See figure 2.

Then the above result has important practical consequences. It shows that for reasoning about safety properties $h(1,2)$ over two indices, we may reduce the model checking problem for an arbitrarily large system $\mathcal{U}$ involving processes P_1 and P_2 to model checking over one of the two systems $\mathcal{V}_{\mathsf{c}}(P_1, P_2)$ or $\mathcal{V}_{\mathsf{nc}}(P_1, P_2)$. Specifically, we have the following

Theorem 3.2 *Given system $\mathcal{U}$ containing P_1 and P_2, define*

$$\mathcal{V} = \begin{cases} \mathcal{V}_{\mathsf{c}}(P_1, P_2) & \textit{if } P_1 \textit{ and } P_2 \textit{ share a common token in } \mathcal{U} \\ \mathcal{V}_{\mathsf{nc}}(P_1, P_2) & \textit{if } P_1 \textit{ and } P_2 \textit{ do not share any common token in } \mathcal{U} \end{cases}$$

Then $\mathcal{U} \models \mathsf{E}_{\mathsf{fin}} h(1,2)$ iff $\mathcal{V} \models \mathsf{E}_{\mathsf{fin}} h(1,2)$.

Note that which of the two alternatives is used for $\mathcal{V}$ can be determined *efficiently* as all we need to do is decide whether in the given system, P_1 and P_2 share a common token or not. This can be done in time $O(|Tok(P_1)| + |Tok(P_2)|) = O(|\mathcal{U}|)$, where $|\mathcal{U}|$ denotes the sum of the sizes of the program texts of all the processes of $\mathcal{U}$.

4 Reduction Results for Fair Infinite Behavior

Given any system $\mathcal{U}$ containing the set $\mathcal{S}$ of processes, define the set of neighbours of $\mathcal{S}$, denoted by $\mathsf{N}(\mathcal{U}, \mathcal{S})$, to be the set of processes $\{X \mid X \notin \mathcal{S} \wedge (\forall S \in \mathcal{S} :$

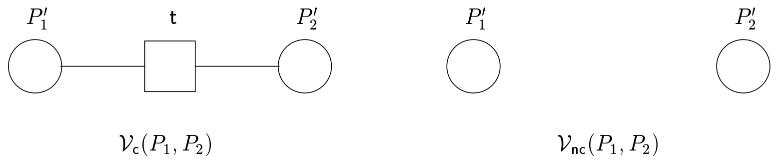

Fig. 2. Two Small Systems

$Tok(X) \cap Tok(S) \neq \emptyset)\}$. An infinite path x of $\mathcal{U}$ is said to be unconditionally fair provided that each of the processes $P_1, ..., P_n$ executes infinitely often in x. We write $\mathsf{E_{uf}}$ and $\mathsf{A_{uf}}$ to quantify purely over unconditionally fair paths of the system.

Theorem 4.1 *Given system $\mathcal{U}$ containing processes P_1 and P_2, let $\mathcal{V}$ be any system containing respectively replica processes P_1' and P_2' such that*

- $Tok(P_1) \cap Tok(P_2) = \emptyset$ *iff* $Tok(P_1') \cap Tok(P_2') = \emptyset$.
- $\mathsf{N}(\mathcal{U}, \{P_1, P_2\}) = \emptyset$ *iff* $\mathsf{N}(\mathcal{V}, \{P_1', P_2'\}) = \emptyset$.

Then $\mathcal{U} \models \mathsf{E_{uf}} h(1,2)$ *iff* $\mathcal{V} \models \mathsf{E_{uf}} h(1,2)$.

Proof Idea ($\Rightarrow$) Given an unconditionally fair path x of $\mathcal{U}$, we will construct an unconditionally fair path y of $\mathcal{V}$ such that x projected onto coordinates 1 and 2 is stuttering equivalent to y projected onto 1 and 2. The construction is similar to the one used in the proof of theorem 3.1, except that whenever process P_1' or process P_2' executes a release transition, we let processes of $\mathcal{V}$ other than P_1' and P_2' that are enabled, execute a transition each in round robin fashion. It is shown in the full paper([13]) that this modification ensures that if x is unconditionally fair, then along the resulting computation path y, all processes of $\mathcal{V}$ execute infinitely often.

($\Leftarrow$) Analogous to the above proof. $\square$

It can be shown that the above result allows us to *efficiently* reduce reasoning for behaviour along unconditionally fair paths from any given system to a systems with at most 3 processes.

5 Potential for Deadlock

We say that process P_i is *individually deadlocked* at location l_i in global state s of system $\mathcal{U}$ provided that there is no future behavior of the system along which P_i succeeds in executing a transition. In our framework, this can be captured in CTL$\backslash$X as AGl_i or in LTL$\backslash$X as just Gl_i. However, the important concept is whether P_i is *deadlockable*, i.e., whether it is *possible* for the system $\mathcal{U}$ to reach from its initial state some such deadlock state for process P_i. This is expressible in CTL$\backslash$X straightforwardly by the truth of $EF \vee_l (AGl_i)$ at the initial state i of $\mathcal{U}$. However, the logic LTL$\backslash$X cannot specify such potential for deadlock. We therefore develop separate methods for analysis of deadlockability.

In our model of computation, process P_i is deadlocked at location l_i in global state s of $\mathcal{U}$, iff (i) at l_i all outgoing transitions are acquire transitions and (ii) there is no future behaviour of the system starting at s along which P_i succeeds in getting all of the tokens in $Tok(P_i)$. Deadlockability of a process in a given system can be inferred from the network topology and a static analysis of the individual processes. A necessary condition for a process P_i to be deadlockable is that it lies on a "lollipop" in the graph of $\mathcal{U}$. A *lollipop* in $\mathcal{U}$ is a cycle or a finite path leading to a cycle. A necessary and sufficient condition for deadlock to occur in a global state is the existence of a "waiting chain" in that state, which may be viewed as an annotated lollipop. Formally, a *waiting chain* for process P_i in reachable global state s is a sequence of processes and tokens of the form $Q_0, t_1, Q_1, t_2, Q_2, ..., Q_j, t_{j+1}, \ldots, t_k, Q_k$ where $k \geq 2$, $P_i = Q_0$, for some $j \in [1 : k-1]$: $Q_j = Q_k$ and for each $i \in [1 : k]$ token t_i is shared by both Q_{i-1} and Q_i, but possessed by Q_i.

For our general model, which permits arbitrary network topologies and heterogeneous processes, the cost of statically analyzing for deadlockability — as a function of the size of the system, which is the size of the network plus the sum of the size of the program texts for individual processes — is potentially high. In fact it can be shown that given system $\mathcal{U}$ and process P_i of $\mathcal{U}$, the problem of deciding whether P_i is deadlockable is NP-complete in the size of the program text of the given system. [1] Thus, in general, there can no efficient reduction from a big system to a small, constant sized system that preserves deadlockability characteristics of processes and their replicas. However for a wide range of potentially useful systems it is possible to give heuristics that reduce the size of a system while preserving the deadlockability characteristics of a given process (or a pair thereof), the discussion of which is deferred to the full paper [13].

In section 6, we give a provably efficient and systematic method of deadlockability analysis for ring structured systems that directly applies to our examples.

6 Applications and Examples

As examples, we consider parameterized formulations of a number of variants of the Dining Philosophers Problem. To our knowledge, our method provides the first fully automatic exact method for verifying parameterized versions of all these variants.

Traditionally, the Dining Philosophers problem is defined in terms of the following informal scenario. There are $n+1$ philosophers $P_0, ..., P_n$, seated around a table in a counterclockwise fashion, *thinking*. Between each pair of philosophers P_i, P_{i+1} is a single fork (token) f_i. From time to time, any philosopher might become *hungry* and attempt to *eat*. In order to eat, the philosopher needs exclusive use of the two adjacent forks. After eating, the philosopher relinquishes the two forks and resumes thinking. Let the resulting system be denoted by $\mathsf{R}(P_0, ..., P_n)$. We will consider solutions to the following variants of the Dining Philosophers problem:

Classic Dining Philosophers - which is the basic, naive solution comprised of deterministic, homogeneous philosophers.

[1] This could still be advantageous from the point of view of parameterized reasoning which is, in general, undecidable.

Nondeterministic Dining Philosophers - which is similar to the Classic, but imposes no restrictions on the order in which forks are acquired.
Asymmetric Dining Philosophers - which introduces a distinguished, dis-similar philosopher to ensure freedom from deadlock.
Right-Left Philosophers - which again involves heterogeneous philosophers. Assuming a ring of even size, the odd indexed philosophers give priority to grabbing their left fork while the even indexed philosophers favor their right.

6.1 Ring Topologies and Cutoffs for Deadlockability

In this section, we first give a systematic, general and efficient test for deadlockability of ring structured systems. We will then use it to provide a uniform treatment of all of the above mentioned Dining Philosophers solutions.

For each process P_i, in the system $\mathsf{R}(P_0, ..., P_n)$, we define its *acquisition type*, denoted by *acquisition-type*(P_i), as a tuple of the form $(k(P_i), o(P_i))$, where

- $k(P_i)$ is the number of blocks into which $Tok(P_i)$ is partitioned, i.e., either 1 or 2, and
- $o(P_i)$ indicates the orderings in which tokens can be acquired and is defined as follows: if $k(P_i) = 1$ then $o(P_i)$ equals $\emptyset$ else if $k(P_i) = 2$ then $o(P_i)$ equals $<$, $>$ or $\emptyset$ respectively if P_i must acquire its left token before its right token, P_i must acquire its right token before its left token and, P_i's left and right tokens could be acquired in any order.

Then *acquisition-type*$(\mathsf{R}(P_0, ..., P_n))$ is defined as the set $\{$*acquisition-type*$(P_i) \mid i \in [0 : n]\}$. We say that processes P_i and P_j each belonging to (possibly different) systems having ring topologies are *acquisition-equivalent* iff they have the same acquisition type. It is easy to see that acquisition-equivalence is an equivalence relation on the set of all processes belonging to systems having ring topologies. Note that if $k(P_j) = 1$, then P_j must have the acquisition type $(1, \emptyset)$; if $k(P_j) = 2$, then it has acquisition type $(2, <)$, $(2, >)$ or $(2, \emptyset)$. Thus the acquisition-equivalence relation partitions the set of all processes belonging to systems with a ring topology into 4 equivalence classes. The following important result establishes a cutoff exactly preserving deadlockability from a large ring to a small ring.

Theorem 6.1 *Given system* $\mathsf{R}(P_0, ..., P_n)$ *where* $n \geq 1$, *let system* $\mathsf{R}(P'_0, ..., P'_m)$ *be such that* $m \geq 1$, P'_0 *is a replica of* P_0 *and acquisition-type*$(\mathsf{R}(P_0, ..., P_n)) =$ *acquisition-type*$(\mathsf{R}(P'_0, ..., P'_m))$. *Then process* P_0 *is deadlockable in* $\mathsf{R}(P_0, ..., P_n)$ *iff* P'_0 *is deadlockable in* $\mathsf{R}(P'_0, ..., P'_m)$.

Proof ($\Rightarrow$) Let process P_0 in $\mathsf{R}(P_0, ..., P_n)$ be deadlockable. Then there is a reachable global state s of $\mathsf{R}(P_0, ..., P_n)$ such that in s either the sequence $P_0, t_0, ..., P_n, t_n$ or the sequence $P_0, t_n, P_n, ..., P_1, t_0$ forms a waiting chain. Assume for definiteness that it is the former. Then since in global state s, each process P_i possesses its left token but not its right token it follows that $k(P_i) = 1$ and $o(P_i)$ equals $<$ or $\emptyset$. But since *acquisition-type*$(\mathsf{R}(P_0, ..., P_n)) =$ *acquisition-type*$(\mathsf{R}(P'_0, ..., P'_m))$, each process in the set $\{P'_0, ..., P'_m\}$ is acquisition-equivalent to a process in $\{P_0, ..., P_n\}$, and so we have that for each process P'_j in $\mathsf{R}(P'_0, ..., P'_m)$, $k(P'_j) = 1$ and $o(P'_j)$ equals $<$ or $\emptyset$. Recall that for each $i \in [0 : m]$, processes P'_i and P'_{i+1} share token t'_i in $\mathsf{R}(P'_0, ..., P'_m)$. Therefore, there is a reachable global state s' of $\mathsf{R}(P'_0, ..., P'_m)$ such that the sequence $P'_0, t'_0, ..., P'_m, t'_m$ forms a waiting chain in s' and so P'_0 is deadlockable in $\mathsf{R}(P'_0, ..., P'_m)$.

($\Leftarrow$) Analogous to the above direction. □

When model checking for safety properties, we have

Corollary 6.2 $\mathsf{R}(P_0, P_1, ..., P_n) \models \mathsf{E_{fin}}h(0, 1)$ *iff* $\mathsf{R}(P'_0, P'_1, ..., P'_m) \models \mathsf{E_{fin}}h(0, 1)$, *where P'_0 and P'_1 are replicas of P_0 and P_1, respectively.*

Proof The result follows immediately from theorem 3.1, by noting that both the sets $Tok(P_0) \cap Tok(P_1) = \{t_0\}$ and $Tok(P'_0) \cap Tok(P'_1) = \{t'_0\}$ are non-empty. □

When model checking while restricting ourselves to unconditionally fair paths, we have

Corollary 6.3 $\mathsf{R}(P_0, P_1, ..., P_n) \models \mathsf{E_{uf}}h(0, 1)$ *iff* $\mathsf{R}(P'_0, P'_1, ..., P'_m) \models \mathsf{E_{uf}}h(0, 1)$, *where $m = n$ if $n \leq 2$ and $m \geq 3$ otherwise, and P'_0 and P'_1 are replicas of P_0 and P_1, respectively.*

Proof The result follows immediately from theorem 4.1, by noting that both the sets $Tok(P_0) \cap Tok(P_1) = \{t_0\}$ and $Tok(P'_0) \cap Tok(P'_1) = \{t'_0\}$ are non-empty and $\mathsf{N}(\mathsf{R}(P_0, P_1, ..., P_n), \{P_0, P_1\})$ and $\mathsf{N}(\mathsf{R}(P'_0, P'_1, ..., P'_m), \{P'_0, P'_1\})$ are both equal to $\emptyset$ if $n = 1 = m$ or $n, m \geq 3$, and equal to $\{P_2\}$ and $\{P'_2\}$, respectively, if $n = 2 = m$. □

Then combining the results in 6.1, 6.2 and 6.3, we have the following

Theorem 6.4 *Given $\mathcal{U} = \mathsf{R}(P_0, ..., P_n)$ where $n \geq 1$, define $\mathcal{V} = \mathsf{R}(P'_0, ..., P'_m)$, where $m = n$ if $n \leq 2$ and $m \geq 3$ otherwise, P'_0 and P'_1 are replicas of P_0 and P_1 respectively, no two processes in $\mathcal{V}$ except possibly P'_0 and P'_1 are acquisition-equivalent and acquisition-type($\mathsf{R}(P_0, ..., P_n)$) = acquisition-type($\mathsf{R}(P'_0, ..., P'_m)$). Then*

- *P_0 and P_1 are deadlockable in $\mathcal{U}$ iff P'_0 and P'_1 are deadlockable in $\mathcal{V}$, respectively*
- *$\mathcal{U} \models \mathsf{E_{fin}}h(0, 1)$ iff $\mathcal{V} \models \mathsf{E_{fin}}h(0, 1)$ and*
- *$\mathcal{U} \models \mathsf{E_{uf}}h(0, 1)$ iff $\mathcal{V} \models \mathsf{E_{uf}}h(0, 1)$.*

Since no two processes among $P'_0, P'_1, ..., P'_m$ in the statement of the above theorem except possibly P'_0 and P'_1 are acquisition-equivalent and *acquisition-type*($\mathsf{R}(P_0, ..., P_n)$) = *acquisition-type*($\mathsf{R}(P'_0, ..., P'_m)$), we have that if $n \geq 3$ then m less than or equal to the maximum of 3 and the cardinality of *acquisition-type*($\mathsf{R}(P_0, ..., P_n)$). Thus we have the following result.

Theorem 6.5 *We can reduce reasoning about the deadlock characteristics, safety properties and liveness properties under the assumption of unconditional fairness for a pair of adjacent processes for the ring* $\mathsf{R}(P_0, ..., P_n)$ *where* $n \geq 3$ *to the ring* $\mathcal{V} = \mathsf{R}(P'_0, ..., P'_m)$*, where* m *less than or equal to the maximum of 3 and the cardinality of acquisition-type*$(\mathsf{R}(P_0, ..., P_n))$.

Since the number of equivalence classes under the acquisition-equivalence relation is 4, therefore the cardinality of the set *acquisition-type*$(\mathsf{R}(P_0, ..., P_n))$ is at most 4, we have

Corollary 6.6 *We can reduce reasoning about the deadlock characteristics, safety properties and liveness properties under assumptions of unconditional fairness for a pair of adjacent processes for an arbitrary ring to a ring of size at most 5.*

6.2 Classical Dining Philosophers

In the classical, "symmetric" solution, the philosophers are homogeneous (run the same program up to renaming), and each philosopher P_i tries first to pick up its left fork and only if it is able to get hold of that does it attempt to pick up its right fork. Thus for each i, we have $\prec_i = \{(f_{i-1}, f_i)\}$. We need to verify that for all adjacent pairs of philosophers P_i and P_{i+1}, the mutual exclusion requirement is met, viz., $\bigwedge_i \mathsf{A_{fin}}\mathsf{G}\neg(eat_i \wedge eat_{i+1})$, and also test whether each of the processes is deadlockable or not. Using a simple symmetry argument (dubbed "state symmetry" in [17]), we see that it is necessary and sufficient to check whether the property $\mathsf{A_{fin}}\mathsf{G}\neg(eat_0 \wedge eat_1)$ is satisfied and whether P_0 is deadlockable. (In short, if $\mathcal{U} \models \mathsf{A_{fin}}\mathsf{G}\neg(eat_0 \wedge eat_1)$, and π is the rotation permutation driving 0 to i, then since $\mathcal{U}$ is symmetric under rotations, $\mathcal{U} \models \mathsf{A_{fin}}\mathsf{G}\neg(eat_i \wedge eat_{i+1})$; and conversely. Similarly to check liveness for philosopher P_i under assumptions of unconditional fairness, viz., whether the property $\mathsf{A_{uf}}\mathsf{G}(hungry_i \Rightarrow \mathsf{F}eat_i)$ is satisfied in suffices to check whether $\mathsf{A_{uf}}\mathsf{G}(hungry_0 \Rightarrow \mathsf{F}eat_0)$ is satisfied. Using theorem 6.5, we have that it is both necessary and sufficient to model check the system comprised of up to 4 philosophers for $\mathsf{A_{fin}}\mathsf{G}\neg(eat_0 \wedge eat_1)$, $\mathsf{A_{uf}}\mathsf{G}(hungry_0 \Rightarrow \mathsf{F}eat_0)$ and for deadlockability of process P_0. We emphasize that this establishes mutual exclusion correctness for philosopher rings of all sizes.

6.3 Non-deterministic Dining Philosophers

In the Non-deterministic Dining Philosophers variant the philosophers are homogeneous, and each philosopher P_i could potentially pick up forks f_{i-1} or f_i in any order. We need to verify that for all adjacent pairs of philosophers P_i and P_{i+1}, the mutual exclusion requirement is met, viz., $\bigwedge_i \mathsf{A_{fin}}\mathsf{G}\neg(eat_i \wedge eat_{i+1})$ and test for the deadlockability of each individual philosopher in the system. Using a simple symmetry argument as before, we see that it is necessary and sufficient to check whether the property $\mathsf{A_{fin}}\mathsf{G}\neg(eat_0 \wedge eat_1)$ is satisfied and that process P_0 is deadlockable. Again, since all philosophers are acquisition-equivalent, we see that it is both necessary and sufficient to model check the system composed of up to 4 philosophers for $\mathsf{A_{fin}}\mathsf{G}\neg(eat_0 \wedge eat_1)$, $\mathsf{A_{uf}}\mathsf{G}(hungry_0 \Rightarrow \mathsf{F}eat_0)$ and deadlockability of P_0.

6.4 Asymmetric Dining Philosophers Problem

To ensure a deadlock free solution the symmetry is broken by making one process, P_0 say, different from the rest of the processes in that P_0 can either acquire both t_0 and t_n if they are free or none at all.The other philosophers could pick up any of the forks, if free, in a non-deterministic fashion. Thus all philosophers P_i, where $i \in [1..n]$ are acquisition-equivalent to each other, with P_0 belonging to a different equivalence class. To check that the mutual exclusion requirement is satisfied it is both necessary to sufficient to check whether the properties $\mathsf{AG}\neg(eat_0 \wedge eat_1)$ and $\mathsf{AG}\neg(eat_1 \wedge eat_2)$ are satisfied. Similarly to test for liveness and deadlockability it suffices to check for the liveness and deadlockability of P_0 and P_1. Then from theorem 6.5, it is necessary and sufficient to check that the systems $\mathsf{R}(P_0, P_1, ..., P_m)$, where $m \leq 3$, satisfies the desired properties.

6.5 Right-Left Dining Philosophers Algorithm

Although the asymmetric nature of the previous problem guarantees a deadlock free solution, there still exists the problem of long waiting chains with each process waiting for a token in possession of the next process in the chain. In the Right-Left Dining Philosophers problem the number of philosophers, n, is even. Each odd philosopher tries first to pick up its left token and only if it is able to acquire it does to proceed to acquire the right token. Each even numbered philosopher, on the other hand, tries first to pick up its right token and only when it is able to acquire it does it proceed to acquire the left token. Therefore $\prec_i$ equals $\{(t_i, t_{i-1})\}$ if i is even and $\{(t_{i-1}, t_i)\}$ if i is odd. Thus any two even philosophers or any two odd philosophers are acquisition equivalent and any odd philosopher and an even philosopher are not acquisition equivalent. Again, using state symmetry, we see that to check that the mutual exclusion requirement is satisfied, it is both necessary and sufficient to check whether the property $\mathsf{AG}\neg(eat_0 \wedge eat_1)$ is satisfied. Furthermore, it suffices to check for the liveness and deadlockability of processes P_0 and P_1 only. Using theorem 6.5, we see that this can be established by checking that a system with up to 4 philosophers, satisfies the requisite properties.

7 Concluding Remarks

In this paper, we have given techniques for limiting state explosion in the context of resource allocation problems. We considered resource allocation systems where processes share tokens representing resources that are needed to proceed. We show that given any such large system organized into a possibly irregular network of n possibly heterogeneous processes, model checking over that system can be efficiently reduced to model checking over a small system. We establish our results for correctness properties expressed in LTL\X, the choice of which was governed by the fact that it is simpler and technically easier to reason about than branching time logics. Furthermore, incorporating the next-time operator X in the temporal logic could enable us to count the number of processes in a given system, thus making it difficult to reduce reasoning of a system with a large number of processes to one with a small number. This justifies exclusion of the next-time operator from our logic. The precise size and topology of the small system is dependent on the large system, as well as the number of indices (processes tracked/observed) in the correctness specification When the network is symmetric and the processes homogeneous our method specializes to provide an efficient solution to the Parameterized Model Checking Problem. As an application we show how to efficiently verify the parameterized Dining Philosophers Problem.

There has been little work on reduction of state explosion for large irregular or heterogeneous systems. Perhaps the works that come closest in spirit to this paper are ([11], [18]) where various notions of approximate symmetry are considered to permit symmetry-like reductions for certain asymmetric systems. However, those methods do not seem to be as broadly applicable to flatly irregular networks of processes as the one proposed here. Most of the work in the literature seems to require highly regular systems with, e.g., homogeneous processes and symmetric or repetitive network topologies. This often makes it possible to directly exploit symmetry (cf. [7], [14], [15], [17], [21]).

PMCP is, in general, undecidable [2]. However, due to the importance of the problem, much effort has been devoted to obtaining (partial) solutions. Under restrictions positive results have been obtained ([9], [21], [22], [27]). Unfortunately most of these methods suffer, first, from the drawback of being only partially automated and hence requiring human ingenuity, and, second, from being sound but not guaranteed complete (i.e., a path "upstairs" maps to a path "downstairs", but paths downstairs do not necessarily lift). Other potentially useful methods can be fully automated but regrettably do not appear to have a clearly defined class of protocols on which they are guaranteed to terminate successfully ([10], [26]). More recently, a proof tree based decision procedure has been proposed in [23] that dealt solely with the parameterized verification for safety properties and handles a wide range of systems in a unified manner, but no complexity bounds were shown. In ([1],[25]), it is shown how to reduce the PMCP for safety for a certain class of systems to systems with a cutoff number of processes. While the results deal effectively with safety properties for a broad range of applications, extensions to properties other than safety were not given. The problem of reasoning about systems comprised of finite but arbitrarily large number indistinguishable processes communicating by rendezvous (two processes exchange a message) was considered by German and Sistla in [19]. They obtain a fully automated and efficient solution to PMCP for single index properties, modulo deadlock. Emerson and Namjoshi [15] show that in a single class (or client-server) synchronous framework PMCP is decidable but with PSPACE-complete complexity. However the closest results are those of Emerson and Namjoshi [14] who for the token ring model show how to reduce reasoning for multi-indexed temporal logic formulae, for rings of arbitrary size to rings of small cutoff size. Their results were formulated for a single process class, a single token and a restricted topology. In [12], Emerson and Kahlon show for systems with conjunctive and disjunctive guards how to reduce reasoning for systems of any size to a small cutoff size with efficient results being obtained for a broad class of properties including safety.

We believe that our results are significant for several reasons. They are applicable to arbitrarily large, heterogeneous, irregular systems; reducing such a system to an equivalent small system. Our methods specialize in the case of regular networks of homogeneous systems to permit efficient parameterized reasoning. Moreover, our methods are fully automated (algorithmic), returning a yes/no answer, are sound and complete, i.e., exact and efficient for many important properties including safety. Our techniques can be extended to the branching time logic CTL*\X, albeit at the cost of greater technical intricacy and computational complexity. Also the model of computation can be generalized in many ways. First we can allow each token to be shared by more than two processes giving us a bipartite graph over processes and tokens as a formal model. Furthermore, we can allow multiple acquire transitions that require different sets of tokens to be grabbed for execution thus allowing us to model, for instance, solutions to the Drinking Philosophers Problem [6].

References

1. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. CAV 2001, LNCS, 2001.
2. K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 15, pages 307-309, 1986.
3. M.C. Browne, E.M. Clarke and O. Grumberg. Reasoning about Networks with Many Identical Finite State Processes. *Information and Control*, 81(1), pages 13-31, April 1989.
4. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*. C-35(8). pp 677-691, August 1986.
5. J.R. Burch, E.M. Clarke, K. L. McMillan, D.L. Dill and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. LICS 1990.
6. K.M. Chandy and J. Misra. The Drinking Philosophers Problem. *ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 4, pp 632-646, 1 984.
7. E.M. Clarke, T. Filkorn and S. Jha. Exploiting Symmetry in Temporal Model Checking. In *Computer Aided Verification, Proceedings of the 5th International Conference*. LNCS 697, Springer-Verlag, 1993.
8. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logics. In *Proceedings of the IBM Workshop on Logics of Programs*, LNCS 131, 1981.
9. E.M. Clarke and O. Grumberg. Avoiding the State Explosion Problem in Temporal Logic Model Checking Algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 294-303, 1987.
10. E.M. Clarke, O. Grumberg and S. Jha. Verifying Parameterized Networks using Abstraction and Regular Languages. In *CONCUR '95: Concurrency Theory, Proceedings of the 6th International Conference*, LNCS 962, pages 395-407, Springer-Verlag, 1995.
11. E. A. Emerson, J. Havlicek and R. Trefler. Virtual Symmetry. LICS 2000.
12. E.A. Emerson and V. Kahlon. Reducing Model Checking of the Many to the Few. In *Automated Deduction - CADE-17*, LNAI 1831, Springer, pages 236-254, 2000.
13. E.A. Emerson and V. Kahlon. Model Checking Large-scale and Parameterized Resource Allocation Systems. Tech. Report, The Univ. of Texas at Austin, 2001.
14. E.A. Emerson and K.S. Namjoshi. Reasoning about Rings. In *Conference Record of POPL '95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 85-94, 1995.
15. E.A. Emerson and K.S. Namjoshi. Automatic Verification of Parameterized Synchronous Systems. In *Computer Aided Verification, Proceedings of the 8th International Conference*. LNCS , Springer-Verlag, 1996.
16. E.A. Emerson and A.P. Sistla. Symmetry and Model Checking. In *Computer Aided Verification, Proceedings of the 5th International Conference*. LNCS 697, Springer-Verlag, 1993.

17. E.A. Emerson and A.P. Sistla. Utilizing Symmetry when Model-Checking under Fairness Assumptions: An Automata-Theoretic Approach. ACM Trans. on Prog. Lang. and Systems (TOPLAS), pp. 617–638, vol. 19, no. 4, July 1997.
18. E. Emerson and R. Trefler. From Asymmetry to Full Symmetry. CHARME99, LNCS, 1999.
19. S.M. German and A.P. Sistla. Reasoning about Systems with Many Processes. *J. ACM*,39(3), July 1992.
20. P. Godefroid and P. Wolper. Using Partial orders for the efficient verification of deadlock-freedom and safety properties. *Formal Methods in Systems Design*. 2(2), pp 149-164, 1993.
21. C. N. Ip and D. Dill, Verifying Systems with Replicated Components in Murphi, pp. 147-158 CAV 1996.
22. R. P. Kurshan and K. L. McMillan. A Structural Induction Theorem for Processes. In *Proceedings of the Eight Annual ACM Symposium on Principles of Distributed Computing*, pages 239-247, 1989.
23. M. Maidl. A Unifying Model Checking Approach for Safety Properties of Parameterized Systems. CAV 2001, LNCS, 2001.
24. D. Peled. Combining partial order reductions with on-the-fly model checking. *Formal Aspects of Computing*, 8, pp 39-64, 1996.
25. A. Pnueli, S. Ruah, and L. Zuck. Automatic Deductive Verification with Invisible Invariants. TACAS 2001, LNCS, 2001.
26. A. P. Sistla. Parameterized Verification of Linear Networks Using Automata as Invariants, CAV, 1997, 412-423.
27. P. Wolper and V. Lovinfosse. Verifying Properties of Large Sets of Processes with Network Invariants. In J. Sifakis(ed) *Automatic Verification Methods for Finite State Systems*, Springer-Verlag, LNCS 407, 1989.