On Reasoning about Rings*

E. Allen Emerson[1] and Kedar S. Namjoshi[2]

[1] Department of Computer Sciences, The University of Texas at Austin
emerson@cs.utexas.edu
[2] Bell Laboratories, Lucent Technologies
kedar@research.bell-labs.com

Abstract. Distributed protocols are often composed of similar processes connected in a unidirectional ring network. Processes communicate by passing a token in a fixed direction; the process that holds the token is allowed to perform certain actions. Usually, correctness properties are expected to hold irrespective of the size of the ring. We show that the question of checking many useful correctness properties for rings of *all* sizes can be reduced to checking them on ring of sizes up to a small *cutoff* size. We apply our results to the verification of a mutual exclusion protocol and Milner's scheduler protocol.

1 Introduction

The ring is a particularly useful way of structuring a system of concurrently executing processes. Well known examples include protocols for mutual exclusion, leader election, scheduling, and the dining philosophers problem. These have two features in common: the individual processes of the ring are isomorphic (i.e., the "code" of one can be transformed into that of another by a simple renaming), and the desired correctness properties are expected to be satisfied by instances of arbitrary size. The protocols are thus *parameterized* by the number of processes. The usual method of verifying that such a parameterized system satisfies a specification is by a deductive, manual proof (cf. [MC 88,MP 94]), which requires considerable ingenuity, and can be done in practice only for reasonably simple protocols.

In this paper, we show that, for *any* system of many isomorphic processes arranged in a unidirectional ring that communicate through a signalling token, a property holds for *every* instance of the system iff it holds for instances up to a *small cutoff* size. Thus, for systems composed of isomorphic finite state processes, a fully automated technique, such as Model Checking [CE 81,QS 82,CES 86] may be applied for the verification of the infinite-state parameterized system.

Correctness properties are expressed using the logic $\mathsf{CTL}^*\backslash\mathsf{X}$ [BCG 89], $\mathsf{CTL}^*\backslash\mathsf{X}$ allows all formulas of CTL^* [EH 86] that do not contain the next-state operator X. Formulas in $\mathsf{CTL}^*\backslash\mathsf{X}$ are insensitive to "stuttering", i.e., repeated occurrences of the same state label or non-observable actions. The exact timing of actions will vary amongst rings of different sizes, hence it is reasonable to make the correctness properties insensitive to this variation.

Our indexed formulation of $\mathsf{CTL}^*\backslash\mathsf{X}$ can express correctness properties of parameterized systems in a simple and natural manner. Let $g(\boldsymbol{i})$, for a set of variables $\boldsymbol{i}$, be a formula in which every proposition and action is indexed by one of the variables in $\boldsymbol{i}$. It follows that, for example, the instances $g(1)$ and $g(2)$ of the formula $g(i)$ are isomorphic up to re-indexing. We consider correctness properties of the form $(\forall \boldsymbol{i} :: g(\boldsymbol{i}))$. For instance, $(\forall i :: g(i))$ (every process satisfies $g(i)$), and $(\forall i, j : i{\neq}j : g(i,j))$ (every distinct pair of processes satisfies

* This work was supported in part by ITR-CCR-020-5483 & NSF grants CCR-009-8141, and SRC Contract No. 2002-TJ-1026. A preliminary version of this paper, "Reasoning About Rings", was presented at the ACM Symposium on Principles of Programming Languages (POPL), 1995. The work of the second author was done while at the University of Texas at Austin.

$g(i,j)$). The quantified variables range over $[0 \ldots n-1]$, where n is the size of the ring instance over which the formula is evaluated. This range is left implicit for conciseness. For instance, mutual exclusion can be expressed as $(\forall i,j : i{\neq}j : \mathsf{AG}\neg(\mathit{critical}_i \ \wedge \ \mathit{critical}_j))$, and eventual access by $(\forall i :: \mathsf{AG}(\mathit{trying}_i \Rightarrow \mathit{critical}_i))$. Our specific results are as follows:

- $(\forall i :: g(i))$ has a cutoff of 2.
- $(\forall i :: g(i,i+1))$ has a cutoff of 3.
- $(\forall i,j : i{\neq}j : g(i,j))$ has a cutoff of 4.
- $(\forall i,j : i{\neq}j : g(i,i+1,j))$ has a cutoff of 5.

Similar results for other quantifier combinations may be derived by the application of our proof techniques. The rotational symmetry of the ring plays an important role in the proofs of these results. It allows us to reduce the check of a formula such as $(\forall i :: g(i))$, which ranges over all possible process indices, to that of one with a particular index, say $g(0)$. We then establish the existence of a correspondence, with respect to the behavior of process 0, between the state transition graphs of an instance of the system with n processes, and one with the number of processes equal to the cutoff. The symmetry then allows us to establish the result. The key to the correspondence proof is the observation that a segment of the ring not containing any observable processes (e.g., for $(\forall i :: g(i))$, a segment not containing process 0) has behaviour similar to that of a single process.

We use a slightly weaker notion of correspondence (bisimulation up to stuttering) rather than the notion (strong bisimulation) that is used in the symmetry reduction techniques of [ES 93,CFJ 93]. This allows us to show the equivalence of large instances to that of cutoff size, which does not hold under strong bisimulation. In fact, ring networks are subject to very little reduction (a factor of n for an instance with n processes) with the techniques of [ES 93,CFJ 93].

The paper is organized as follows. Section 2 defines the notation and constructions we need. Section 3 contains the definition of the system model. We prove the main results in Section 4. Section 5 contains applications of these results to two protocols. In Section 6 we show a sharp boundary for the decidability of parameterized verification: the question is undecidable for ring networks where the token may carry a single bit of mutable information. Section 7 concludes the paper with a discussion of related work and future extensions. The Appendix contains detailed proofs of the main results.

2 Preliminaries

This section contains definitions of the syntax of the temporal logics considered in the paper, the types of structures over which the logics are interpreted, and the semantics of these logics.

2.1 Notation

The notation follows the format introduced by Dijkstra and Scholten in [DS 90]. Quantified expressions are written in the format $(\mathbf{Q}x : r : p)$, where $\mathbf{Q}$ is the quantifier (e.g., $\exists, \forall, \mathit{min}, \mathit{max}$), x the bound variable, r the range, and p the expression being quantified. When the range of x is clear from the context, it may be dropped, in which case the quantified expression has the form $(\mathbf{Q}x :: p)$. Sets defined by set comprehension are written in the format $\{x | P(x)\}$, which represents the set of all elements x for which the predicate P holds. The power set of a set S is denoted by 2^S. For a binary relation R, we often write $s \ R \ t$ instead of $(s,t) \in R$ for readability. Disjoint set-union is represented by $\uplus$. For $W = S \uplus T$, $W|_S$ $(W|_T)$ is the projection of W on to S (T).

Proofs are often given in the calculational style [DS 90]. A proof in this format is constructed out of the following basic unit:

$$\begin{array}{l} \quad P \\ op\ (\ \text{hint why } P\ op\ Q\ ) \\ \quad Q \end{array}$$

The binary operator op is reflexive and transitive, which allows the chaining together (using transitivity) of a number of similar proof units. Typical operators are $\Rightarrow$ (implies), $\Leftarrow$ (follows-from) and $\Leftrightarrow$ (if-and-only-if).

2.2 Labeled Transition Systems

The types of structures over which temporal logics are interpreted are called Labeled Transition Systems (LTS's) [Kel 76]. Informally, a LTS is a directed graph, where each vertex (a state) is labeled with the atomic propositions true at that state, and each edge (an action) is labeled with an action symbol.

Definition 1 (LTS) *A Labeled Transition System (or LTS) A is a tuple* $(Q, \Sigma, \delta, \lambda, L, I)$ *where*

- Q *is a nonempty set of states,*
- Σ *is a transition (or action) alphabet which contains the distinguished symbol* τ *(the silent action),*
- $\delta \subseteq Q \times \Sigma \times Q$ *is the transition relation,*
- $\lambda : Q \to L$ *is a labelling function on the states,*
- L *is a nonempty set of labels,*
- $I \subseteq Q$ *is the set of initial states.*

For clarity, we write $s \xrightarrow{a}_A t$ instead of $(s, a, t) \in \delta$. We write $s \to_A t$ for $(\exists a : a \in \Sigma : s \xrightarrow{a}_A t)$. The subscript A is often elided when a specific LTS is under consideration. A *path* p of A is a pair (σ, α), where σ and α are sequences (finite or infinite) of states and actions respectively such that $1 + |\alpha| = |\sigma|$, where for every i such that $i + 1 < |\sigma|$, $\sigma_i \xrightarrow{\alpha_i}_A \sigma_{i+1}$. A *fullpath* is a maximal path; i.e., either it is infinite or the last state has no successor. A *computation* is a path where the state sequence starts at an initial state of A. Let the length of a path $p = (\sigma, \alpha)$, $|p|$, equal $|\sigma|$.

The semantics of most temporal logics do not take edge labels into account. They are defined over a special kind of LTS known as a Kripke structure.

Definition 2 (Kripke Structure) *A Kripke Structure is an LTS of the form* $(Q, \{\tau\}, \delta, \lambda, 2^{AP}, I)$*, where* AP *is a set of* atomic propositions.

2.3 LTS Composition

The parallel composition of two LTS's $A = (Q_A, \Sigma_A, \delta_A, \lambda_A, L_A, I_A)$ and $B = (Q_B, \Sigma_B, \delta_B, \lambda_B, L_B, I_B)$ is defined with respect to a bijective partial function $\Gamma : \Sigma_A \backslash \{\tau\} \to \Sigma_B \backslash \{\tau\}$ which defines the synchronized moves of A and B. The composition, denoted by $A \parallel_\Gamma B$, defines the LTS $AB = (Q_{AB}, \Sigma_{AB}, \delta_{AB}, \lambda_{AB}, L_{AB}, I_{AB})$:

1. $Q_{AB} = Q_A \times Q_B$,
2. $\Sigma_{AB} = (\Sigma_A \backslash domain(\Gamma)) \uplus (\Sigma_B \backslash range(\Gamma))$,
3. δ_{AB} is defined by: $(s, t) \xrightarrow{a}_{AB} (u, v)$ iff
   (a) $a \notin domain(\Gamma) \wedge s \xrightarrow{a}_A u \wedge t = v$ (a move by A only), or
   (b) $a \notin range(\Gamma) \wedge t \xrightarrow{a}_B v \wedge s = u$ (a move by B only), or
   (c) $a = \tau \wedge (\exists b : b \in domain(\Gamma) : s \xrightarrow{b}_A u \wedge t \xrightarrow{\Gamma(b)}_B v)$ (a synchronized move)
4. $\lambda_{AB}(s, t) = \lambda_A(s) \uplus \lambda_B(t)$,
5. $L_{AB} = L_A \uplus L_B$, and
6. $I_{AB} = I_A \times I_B$.

It is easily shown that $\parallel_\Gamma$ is associative. It is also symmetric up to strong bisimulation.

2.4 LTS Projection

For an LTS $C = A \parallel_\Gamma B$, the projection of C on A elides the labels on actions performed by the B-component, replacing them with the silent action τ. Formally, the projection is defined as the LTS $C|_A = (Q, \Sigma, \delta, \lambda, L, I)$, where

1. $Q = Q_C$,
2. $\Sigma = \Sigma_A$,
3. δ is defined by: $s \xrightarrow{a} t$ iff
   (a) $a \in \Sigma_A \ \wedge \ s \xrightarrow{a}_C t$, or
   (b) $a = \tau \ \wedge \ (s \xrightarrow{\tau}_C t \ \vee \ (\exists b : b \in (\Sigma_B \backslash \Sigma_A) : s \xrightarrow{b}_C t))$
4. $\lambda(s) = (\lambda_C(s))|_{L_A}$,
5. $L = L_A$, and
6. $I = I_C$

2.5 Temporal Logics and their Semantics

We will define the syntax and semantics of the *Full Branching Time Logic* CTL^*. Other logics such as $\mathsf{CTL}^*\backslash\mathsf{X}$, CTL and LTL may be defined as sub-logics of CTL^*.

CTL^* (read as "Computation Tree Logic - star") [EH 86] derives its expressive power from the freedom of combining modalities which quantify over paths and the modalities which quantify states along a particular path. These modalities are $\mathsf{A}, \mathsf{E}, \mathsf{F}, \mathsf{G}, \mathsf{X}_s$, and U_w ("for all futures", "for some future", "sometime", "always", "strong next-time", and "weak until", respectively), and they are allowed to appear in virtually arbitrary combinations. Usually, CTL^* is interpreted over LTS's where there are no visible actions. For our purposes, however, it is more convenient to consider LTS's with visible actions; hence, we add the operators X_s^a, for each visible action a, obtaining a logic that we call $\mathsf{CTL}^*_{\mathsf{\Sigma}}$. Formally, we inductively define a class of state formulas (true or false of states) and a class of path formulas (true or false of paths), which is the least set of formulas satisfying the following.

(S1) Any atomic proposition P is a state formula.
(S2) If f, g are state formulas, then so are $f \ \wedge \ g$, $\neg f$.
(S3) If f is a path formula then $\mathsf{E}f$ is a state formula.
(P1) Any state formula p is also a path formula.
(P2) If f, g are path formulas, then so are $f \ \wedge \ g$, $\neg f$.
(P3) If f, g are path formulas then so are $\mathsf{X}_s f$, $\mathsf{X}_s^a f$ and $f \mathsf{U}_w g$.

The semantics of a formula is defined with respect to a LTS $M = (S, \Sigma, \delta, \lambda, 2^{AP}, I)$. We write $M, s \models f$ ($M, x \models f$) to mean that state formula f (path formula f) is true in LTS M at state s (of fullpath x, resp.). When M is understood, we write simply $s \models f$ ($x \models f$). We define $\models$ inductively using the convention that x denotes a fullpath and x^i denotes the suffix fullpath starting at the ith state in x, provided $i < |x|$. For a state s:

(S1) $s \models P$ iff $P \in \lambda(s)$, for atomic proposition P
(S2) $s \models f \ \wedge \ g$ iff $s \models f$ and $s \models g$,
$s \models \neg f$ iff not ($s \models f$)
(S3) $s \models \mathsf{E}f$ iff for some fullpath x starting at s, $x \models f$

For a fullpath x:

(P1) $x \models f$ iff $s \models f$, for any state formula f, where s is the first state on p,
(P2) $x \models f \wedge g$ iff $x \models f$ and $x \models g$,
$x \models \neg f$ iff not ($x \models f$)
(P3) $x \models \mathsf{X}_s f$ iff x^1 is defined and $x^1 \models f$,
$x \models \mathsf{X}_s^a f$ iff there exists $i < |x|$ such that for every $j < i$, the jth action on x is a τ-action, the ith action is a, x^{i+1} is defined and $x^{i+1} \models f$,
$x \models (f \,\mathsf{U}_w\, g)$ iff for all $i < |x|$, either for some $j \leq i$, $x^j \models g$, or $x^i \models f$

We say that state formula f is *valid*, and write $\models f$, if, for every LTS M and every state s in M, $M, s \models f$. State formula f is *satisfiable* iff for some LTS M and some state s in M, $M, s \models f$. In this case we also say that M defines a *model* of f. We define validity and satisfiability for path formulas similarly.

Other connectives can then be defined as abbreviations in the usual way: $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $f \Rightarrow g \equiv \neg f \vee g$, $f \Leftrightarrow g \equiv (f \Rightarrow g) \wedge (g \Rightarrow f)$, $\mathsf{A}f \equiv \neg\,\mathsf{E}\,\neg\, f$, $\mathsf{G}f \equiv f \,\mathsf{U}_w\, \mathit{false}$, and $\mathsf{F}f \equiv \neg\,\mathsf{G}\,\neg\, f$. Further operators may also be defined as follows:
$\mathsf{X}_w f \equiv \neg \mathsf{X}_s \neg f$ is the weak next-time,
$f \mathsf{U}_s g \equiv (f \mathsf{U}_w g) \wedge \mathsf{F}g$ is the strong until,
$\overset{\infty}{\mathsf{F}} f \equiv \mathsf{GFX}_s f$ means infinitely often f,
$\overset{\infty}{\mathsf{G}} f \equiv \mathsf{FGX}_s f$ means almost everywhere f.

Sub-logics of CTL^*_Σ are defined either by restrictions on the operators allowed, or by restrictions on the ways in which the operators can be combined. LTL (Linear Temporal Logic) is obtained by leaving out (S2) and (S3), and redefining $M, s \models f$ to be true iff for every fullpath x that starts at s, $M, x \models f$. $\mathsf{CTL}^*_\Sigma \backslash \mathsf{X}$ is obtained by leaving out the operator X_s, while retaining the X^a_s operators. This makes the logic insensitive to *stuttering* (repetition of the same state or of unobservable actions). The paper by Emerson [Em 90] contains a detailed survey of these logics and a comparison of their expressive powers.

2.6 Block Bisimulation

In order to compare different instances of the parameterized ring system, we use a form of bisimulation that (i) ignores stuttering and (ii) compares state propositions in different size instances. We define "block bisimulation", which (i) ignores stuttering by comparing blocks, or sub-paths, where there is no observable change, and (ii) compares different instances w.r.t. functions that map propositions and actions of one instance to the other. It can be shown that on Kripke Structures block bisimulation is equivalent to stuttering bisimulation [BCG 88].

A block bisimulation on a LTS $A = (Q, \Sigma, \delta, \lambda, L, I)$ is given by a relation R on Q, given as a union of a set of relations R_i. Each R_i has associated mapping functions h_i^L and h_i^Σ that map propositions and actions, respectively. For R_0, h_0^L and h_0^Σ are both the respective identity functions.

A block starting at state s is a path from s for which there is no observable change, in the sense that every transition except possibly the final one is a τ-transition, and every state on the path, except possibly the final state is related to s by R_0. The set of finite (infinite) blocks from s is denoted by $\mathit{Fin}_R(s)$ ($\mathit{Inf}_R(s)$). For a finite path $p = (\sigma; v, \alpha; a)$ $\mathit{endS}(p) = v$ and $\mathit{endA}(p) = a$.

Definition 3 (Block Bisimulation) *A relation $R = \{R_i\}$ on Q is a block bisimulation on the LTS A iff*

1. *R_0 is an equivalence relation and $R_0; R_i; R_0 \subseteq R_i$ for every i,*
2. *For any i such that $s \; R_i \; t$,*
   (a) $h_i^L(\lambda(s)) = \lambda(t)$,

(b) $(\forall p : p \in Fin_R(s) : (\exists q : q \in Fin_R(t) : h_i^{\Sigma}(endA(p)) = endA(q) \ \wedge \ endS(p)\ R_i\ endS(q)))$
(c) $(\forall p : p \in Inf_R(s) : (\exists q : q \in Inf_R(t)))$

$\square$

For Kripke Structures, where $L = 2^{AP}$, for any formula f, let $h_i(f)$ be the formula obtained by replacing each occurrence of each proposition P of f by $h_i^L(P)$ and of each action a by $h_i^{\Sigma}(a)$. The following theorem follows from the equivalence of block bisimulation and stuttering bisimulation and the corresponding result in [BCG 88].

Theorem 1 *Let R be a block bisimulation on LTS A. For $s\ R_i\ t$ and any formula f of $\mathsf{CTL}^*_{\Sigma}\backslash\mathsf{X}$, $A, s \models f$ iff $A, t \models h_i(f)$.* $\square$

We write $s \sim t$ (read as "s and t are block equivalent") if there is a block bisimulation on A that includes the pair (s, t). We write $A \sim B$, for LTS's A and B iff there is a block bisimulation on the disjoint union of A and B such that every initial state in A is related to some initial state of B and vice-versa.

2.7 Group Theoretic notions

Some simple notation from Group Theory is needed to define the symmetries of LTS's. For an natural number n, let $[n]$ represent the set $\{x \mid 0 \leq x \leq n-1\}$. Let $Sym\ I$ be the full symmetry group on the index set $I \subseteq \mathbf{N}$, and $\mathcal{C}_n$ be the permutation group of rotations on a ring of size n. For groups $\mathcal{C}$ and $\mathcal{D}$, we write $\mathcal{C} \preceq \mathcal{D}$ to mean that $\mathcal{C}$ is a subgroup of $\mathcal{D}$. Clearly, $\mathcal{C}_n \preceq Sym\ [n]$.

For a formula f indexed over a set I, $Aut\ f = \{\pi \in Sym\ I \mid \pi(f) \equiv f\}$. For a LTS M composed of n processes in parallel, the states are indexed by $[n]$, and $Aut\ M$ is the group of permutations in $Sym\ [n]$ that when applied to every state and transition of M, map M to an isomorphic copy [ES 93]. Abusing notation slightly, we define $Aut\ s$, for a state s, to be the set of permutations that, when applied to s, map it to itself.

3 System Model

An individual process K_i of a parameterized instance is constructed from a "template" process T by indexing the states, actions, and labels of T by i. We require that the template process satisfy the following properties.

- The actions of T are divided into token transfer actions *snd* (send) and *rcv* (receive), and other actions Σ. Of the actions in Σ, some are enabled only if the process has the token, while the others may be enabled without the process possessing the token.
- The first non-Σ action to be executed from the initial state must be a *rcv* action.
- Every *rcv* action is eventually followed by a *snd* action. The template process T may satisfy this condition by construction, or under some local fairness assumption.

For finite-state template processes, these conditions can be checked automatically using a model checker.

The instance of the parameterized system with n processes is defined by $Ring_n = K_0 \parallel K_1 \parallel \ \ldots\ \parallel K_{n-1} \parallel D_n$, where D_n is the initial token distribution process that synchronizes initially with a non-deterministically chosen K_i to pass it the token. For each composition $K_i \parallel K_{i+1}$, the synchronized actions are snd_{i+1} and rcv_{i+1} (all arithmetic in a ring of size n is done modulo n). Let M_n denote the LTS induced by $Ring_n$. For a set of indices $I \subseteq [n]$, we let $M_n|_I$ denote the projection of M_n on to the processes indexed by I. For a set P of propositions indexed by $[n]$, P_I denotes the subset that is indexed with elements from I.

Definition 4 (Ring Intervals) *Let $[i:j]_n$ denote the indices on the clockwise segment from i to j on a ring of size n. Let $[i:j)_n = [i:j]_n - \{j\}$, $(i:j]_n = [i:j]_n \backslash \{i\}$, and $(i:j)_n = [i:j]_n \backslash \{i,j\}$. Note that $(i:i)_n$, $[i:i)_n$, and $(i:i]_n$ are all empty.* $\square$

4 Property specific abstractions

We show here that for specific types of properties, it suffices to consider instances of size at most a small *cutoff* size in order to show that the property holds for *all* instances.

The properties that we consider are of the forms $(\forall i :: g(i))$, $(\forall i, j : i \neq j : g(i, j))$, and $(\forall i, j : i \neq j : g(i, i+1, j))$. In each case, we first show by symmetry arguments, that in each instance of size n, it is sufficient to instantiate the quantifier $(\forall i)$ with some process index, say 0. We then prove that an instance of size greater than the cutoff size satisfies this smaller formula iff the instance of cutoff size does so. This is shown by exhibiting a block bisimulation between the large and the small system, using the following key theorem.

Theorem 2 (Reduction Theorem) *Let $I \subseteq [n]$ and $J \subseteq [k]$ be sets of indices. Let $h : I \rightarrow J$ be a bijection such that for any $i, j \in I$,*

1. *$i \leq j$ iff $h(i) \leq h(j)$, and*
2. *$(i : j)_n \neq \emptyset$ iff $(h(i) : h(j))_k \neq \emptyset$.*

Then $M_n|_I \sim M_k|_J$.

Proof Sketch.

We define first a relation R_{ij} between states of M_i and M_j w.r.t. a bijection $f : X \rightarrow Y$ that satisfies conditions analogous to those satisfied by the function h above. We then instantiate i, j and f in various ways to form a relation between $M_n|_I$ and $M_k|_J$. In the following, let $tok_l.s$ be the index of the process in M_l that possesses the token in state s.

From f, one can derive the renaming function f^A_{ij} for indexed actions and the renaming function f^P_{ij} for indexed propositions as follows: $f^A_{ij}(a_l) = a_{f(l)}$ and $f^P_{ij}(P_l) = P_{h(l)}$. Let R_{ij} be the relation between states of $M_i|_X$ and $M_j|_Y$, defined w.r.t. f as follows.

$s\ R_{ij}\ t$ iff

1. The local states of D_i in s and D_j in t are identical (recall that D is the initial token distribution process),
2. The state of the processes indexed by X in s is identical to the state of the processes indexed by Y in t, up to f: $f^P_{ij}(\lambda(s)|_X) = \lambda(t)|_Y$,
3. The token is at a process l in s iff it is at the corresponding process $f(l)$ in t: $(\forall l : l \in X : tok_i(s) = l \equiv tok_j(t) = f(l))$, and
4. The token is between processes l and m in s iff it is between the corresponding processes in t: $(\forall l, m : l, m \in X : tok_i(s) \in (l : m)_i \equiv tok_j(t) \in (f(l) : f(m))_j)$.

We instantiate this definition in the following ways. Let B_0 be the union of the relations R_{nn} and R_{kk} obtained by taking f to be the identity on I and J respectively. Let B_1 be the relation R_{nk} obtained by instantiating f with the function h in the hypothesis. Let B_2 be the relation R_{kn} obtained by instantiating f with the function h^{-1}. Finally, let $B = B_0 \cup B_1 \cup B_2$. It is straightforward to check that B_0 is an equivalence relation and that $B_0; B_i; B_0 \subseteq B_i$, for all i. The Appendix contains the complete proof that B is a block bisimulation. $\square$

4.1 Properties of the form $(\forall i :: g(i))$

Properties of the form $(\forall i :: g(i))$ refer to properties that every individual process in a system of processes must satisfy. They are typically used to specify progress requirements, as in absence of starvation $(\forall i :: \mathsf{AG}(\mathit{trying}_i \Rightarrow \mathit{critical}_i))$. We show first that for a symmetric system, it is possible to reduce this property to its instantiation with a single index. The following lemma is a refinement of one in [ES 93].

Lemma 1 *If A is the LTS of a system with n isomorphic processes, $\mathcal{C}_n \preceq Aut\ \ A$, and the start state q^0 is symmetric (i.e $Aut\ \ q^0 = Sym\ [n]$) then, $A, q^0 \models (\forall i :: g(i))$ iff $A, q^0 \models g(0)$.*

Proof.

$(LHS \Rightarrow RHS)$ This is immediate from the definition of $\models$.

$(LHS \Leftarrow RHS)$

$$
\begin{aligned}
& \quad A, q^0 \models g(0) \\
\Rightarrow & \ (\ \pi \text{ is a permutation }) \\
& \quad (\forall \pi : \ \pi \in Aut\ \ A : \ \pi(A), \pi(q^0) \models \pi(g(0))) \\
\Leftrightarrow & \ (\ \pi(A) = A \text{ as } \pi \in Aut\ \ A;\ \pi(q^0) = q^0 \text{ as } q^0 \text{ is symmetric }) \\
& \quad (\forall \pi : \ \pi \in Aut\ \ A : \ A, q^0 \models g(\pi(0)) \\
\Rightarrow & \ (\text{ as } \mathcal{C}_n \leq Aut\ \ A\ ) \\
& \quad (\forall \pi : \ \pi \in \mathcal{C}_n : \ A, q^0 \models g(\pi(0)) \\
\Rightarrow & \ (\text{ logic }) \\
& \quad A, q^0 \models (\forall \pi : \ \pi \in \mathcal{C}_n : g(\pi(0)) \\
\Rightarrow & \ (\text{ The cyclic shifts in } \mathcal{C}_n \text{ drive 0 to } i \text{ for every } i \in [n]\ ) \\
& \quad A, q^0 \models (\forall i :: g(i))
\end{aligned}
$$

□

Theorem 3 *Let f be a formula of the form $(\forall i :: g(i))$, where $g(i)$ is a $\mathsf{CTL}^*_{\mathsf{\Sigma}} \backslash \mathsf{X}$ formula that refers only to propositions indexed by i. Then, $(\forall n : n \geq 2 : \ M_n, q_n^0 \models f \ \equiv \ M_2, q_2^0 \models f)$.*

Proof. For any $n \geq 2$,

$$
\begin{aligned}
& \quad M_n, q_n^0 \models f \\
\Leftrightarrow & \ (\text{ the initial state is symmetric, Lemma 1 }) \\
& \quad M_n, q_n^0 \models g(0) \\
\Leftrightarrow & \ (\text{ as } g(0) \text{ refers only to propositions indexed by 0 }) \\
& \quad M_n|_{\{0\}}, q_n^0 \models g(0) \\
\Leftrightarrow & \ (\text{ By Theorem 2, } M_n|_{\{0\}} \sim M_2|_{\{0\}} \text{ w.r.t. } h : 0 \mapsto 0) \\
& \quad M_2|_{\{0\}}, q_2^0 \models g(0) \\
\Leftrightarrow & \ (\text{ as } g(0) \text{ refers only to propositions indexed by 0 }) \\
& \quad M_2, q_2^0 \models g(0) \\
\Leftrightarrow & \ (\text{ by Lemma 1 }) \\
& \quad M_2, q_2^0 \models f
\end{aligned}
$$

□

4.2 Properties of the form $(\forall i :: g(i, i+1))$

Formulas with the form $(\forall i :: g(i, i+1))$ can express properties of neighboring pairs of processes. For instance, the exclusion property for the Dining Philosophers problem may be expressed as $(\forall i :: \mathsf{AG}\neg(\mathit{eating}_i \ \wedge \ \mathit{eating}_{i+1}))$. By a proof analogous to that for Lemma 1, we have the following lemma.

Lemma 2 *If A is the global state transition graph of a system with n isomorphic processes, $\mathcal{C}_n \preceq Aut\ \ A$, and the start state q^0 is symmetric, then $A, q^0 \models (\forall i :: g(i, i+1))$ iff $A, q^0 \models g(0, 1)$.* □

We have to take into account the various situations that process K_0 and K_1 can be in. Intuitively, either K_0 or K_1 can be given the token initially, or some other process could be assigned the token. This suggests that a three process system may be sufficient. And in fact we have, by an argument analogous to that for Theorem 3,

Theorem 4 *Let f be a formula of the form $(\forall i :: g(i, i+1))$, where $g(i, i+1)$ is a $\mathsf{CTL}^*_\Sigma \backslash \mathsf{X}$ formula. Then, $(\forall n : n \geq 3 : M_n, q^0_n \models f \equiv M_3, q^0_3 \models f)$* $\square$

4.3 Properties of the form $(\forall i, j : i \neq j : g(i, j))$

We consider next formulas of the form $(\forall i, j : i \neq j : g(i, j))$. These are used to express properties that involve distinct pairs of processes. Mutual exclusion, for instance, can be written as $(\forall i, j : i \neq j : \mathsf{AG}\neg(\mathit{critical}_i \ \wedge \ \mathit{critical}_j))$.

Lemma 3 *If A is the LTS of a system with n isomorphic processes, $\mathcal{C}_n \preceq Aut \ \ A$, and the start state q^0 is symmetric then, $A, q^0 \models (\forall i, j : i \neq j : g(i, j))$ iff $A, q^0 \models (\forall j : j \neq 0 : g(0, j))$.*

Proof.

$(LHS \Rightarrow RHS)$ This is immediate from the definition of $\models$.

$(LHS \Leftarrow RHS)$

$$
\begin{array}{l}
\quad A, q^0 \models (\forall j : j \neq 0 : g(0, j)) \\
\Rightarrow (\ \pi \text{ is a permutation }) \\
\quad (\forall \pi :\ \pi \in Aut \ \ A :\ \pi(A), \pi(q^0) \models \pi(\forall j : j \neq 0 : g(0, j))) \\
\Leftrightarrow (\ \pi(A) = A \text{ as } \pi \in Aut \ \ A,\ \pi(q^0) = q^0, \text{ as } q^0 \text{ is symmetric }) \\
\quad (\forall \pi :\ \pi \in Aut \ \ A :\ A, q^0 \models (\forall j : j \neq \pi(0) : g(\pi(0), j))) \\
\Rightarrow (\text{ as } \mathcal{C}_n \leq Aut \ \ A\ ) \\
\quad (\forall \pi :\ \pi \in \mathcal{C}_n :\ A, q^0 \models (\forall j : j \neq \pi(0) : g(\pi(0), j))) \\
\Rightarrow (\text{ logic }) \\
\quad A, q^0 \models (\forall \pi :\ \pi \in \mathcal{C}_n :\ (\forall j : j \neq \pi(0) : g(\pi(0), j))) \\
\Rightarrow (\text{ The cyclic shifts in } \mathcal{C}_n \text{ drive } 0 \text{ to } i \text{ for every } i \in [n]\ ) \\
\quad A, q^0 \models (\forall i, j : i \neq j : g(i, j))
\end{array}
$$

$\square$

From this lemma, we need to consider only the pairs of processes $(0, j)$, such that $j \neq 0$. From the Reduction Theorem, we obtain the following theorem.

Theorem 5 *For any $n \geq 4$,*

1. $M_n|_{(0,1)} \sim M_4|_{(0,1)}$,
2. $M_n|_{(0,n-1)} \sim M_4|_{(0,3)}$, *and*
3. $M_n|_{(0,j)} \sim M_4|_{(0,2)}$, *for* $j \notin \{0, 1, n-1\}$.

Proof. Let $h : [n] \to [4]$ be defined by $h(0) = 0$, $h(1) = 1$, $h(n-1) = 3$, and $h(j) = 2$, for $j \notin \{0, 1, n-1\}$. Each of the claims above follows from Theorem 2 by observing that h restricted to the pairs mentioned in each claim is a bijection that satisfies both preconditions of the theorem. The function h may be represented pictorially as shown in Figure 1. $\square$

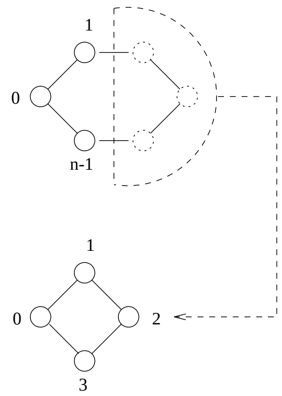

Fig. 1. The abstraction used in the proof of Theorem 5

Theorem 6 *Let f be a formula of the form $(\forall i, j : i{\neq}j : g(i,j))$, where $g(i,j)$ is a $\mathsf{CTL}^*_\Sigma \backslash \mathsf{X}$ formula. Then, $(\forall n : n \geq 4 : M_n, q_n^0 \models f \;\equiv\; M_4, q_4^0 \models f)$.*

Proof. For any $n \geq 4$,

$M_n, q_n^0 \models f$
$\Leftrightarrow$ (the initial state is symmetric, Lemma 3)
$M_n, q_n^0 \models (\forall j : j{\neq}0 : g(0,j))$
$\Leftrightarrow$ (splitting up the formula)
$M_n, q_n^0 \models g(0,1) \;\wedge\; M_n, q_n^0 \models g(0, n-1) \;\wedge$
$M_n, q_n^0 \models (\forall j : j{\notin}\{0,1,n-1\} : g(0,j))$
$\Leftrightarrow$ (by Theorems 5 and 1)
$M_4, q_4^0 \models g(0,1) \;\wedge\; M_4, q_4^0 \models g(0,3) \;\wedge$
$M_4, q_4^0 \models g(0,2)$
$\Leftrightarrow$ (recombining the formula)
$M_4, q_4^0 \models (\forall j : j{\neq}0 : g(0,j))$
$\Leftrightarrow$ (by Lemma 3)
$M_4, q_4^0 \models f$
□

As a particular case, the formula $(\forall i, j : i{\neq}j : \mathsf{AG}\neg(critical_i \;\wedge\; critical_j))$, which expresses mutual exclusion, can be checked in a 4-process system.

4.4 Properties of the form $(\forall i, j : i{\neq}j : g(i, i+1, j))$

Properties of the form $(\forall i, j : i{\neq}j : g(i, i+1, j))$ are used to express global properties that indicate the relationship between a pair of neighboring processes and an arbitrary process. An example of such a property is presented in Section 5. By proofs similar to those for Theorems 3 and 6, we obtain the following theorems.

Lemma 4 *If A is the LTS of a system with n isomorphic processes, $n > 1$. $\mathcal{C}_n \preceq Aut\;\; A$, and the start state q^0 is symmetric, then, $A, q^0 \models (\forall i, j : i{\neq}j : g(i, i+1, j))$ iff $A, q^0 \models (\forall j : j{\neq}0 : g(0,1,j))$.* □

Theorem 7 *Let f be a formula of the form $(\forall i, j : i{\neq}j : g(i, i+1, j))$, where $g(i, i+1, j)$ is a $\mathsf{CTL}^*_\Sigma \backslash \mathsf{X}$ formula. Then, $(\forall n : n \geq 5, M_n, q_n^0 \models f \;\equiv\; M_5, q_5^0 \models f)$.* □

5 Applications

To illustrate the use of the results, we look at two protocols, one for distributed mutual exclusion [WL 89] and Milner's scheduler protocol [Mil 90]. Milner's protocol is known to have an exponential growth in reachable states w.r.t. the number of processes, hence it is often used as a benchmark for model checking systems. Our results show that it suffices to check correctness for instances of size up to 5 in order to conclude correctness for every instance; thus avoiding the problem posed by the exponentially growing reachable state space.

5.1 Distributed Mutual Exclusion

The template process for the protocol in [WL 89] is given below.

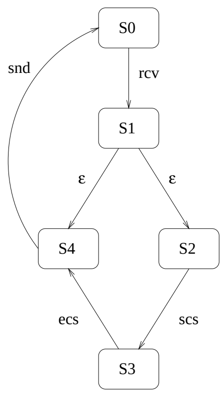

Fig. 2. Template process for the Mutual Exclusion Protocol

Initially, every process is in state $S0$. Here ϵ is used to indicate a local action. It is easily checked that the process description satisfies the conditions for the token passing model (cf. Section 3). So properties such as the following can be checked using a 2-process system by Theorem 3.

- If a process requests the token, it will eventually receive it. $(\forall i :: \mathsf{AG}(S0_i \Rightarrow \mathsf{AF}(S1_i)))$
- Every trying process eventually enters its critical section. $(\forall i :: \mathsf{AG}(S0_i \Rightarrow \mathsf{AF}(S3_i \ \vee \ S4_i)))$

Mutual exclusion, i.e $(\forall i, j : i{\neq}j : \mathsf{AG}\neg(S3_i \ \wedge \ S3_j))$ can be checked in a 4-process system by Theorem 6.

5.2 Milner's scheduler protocol

The template process for the protocol in [Mil 90] is given below:

The initial state is $S3$. To express the correctness properties concisely, we abbreviate $\mathsf{X}_s^{a_i} \mathit{true}$ by $\hat{a}_i$. The correctness properties are as follows.

- Along every computation, $a_0, a_1 \ldots a_{n-1}$ must be performed cyclically, starting with whichever process is enabled first. The following formula expresses the cyclic order. This has a cutoff of 5, by theorem 7.
  $(\forall i, j : i{\neq}j : \mathsf{AG}\ (\hat{a}_i \Rightarrow \mathsf{X}_s^{a_i}((\neg\hat{a}_i \ \wedge \ \neg\hat{a}_j) \ \mathsf{U} \ \hat{a}_{i+1})))$

- Every process performs a_i and b_i alternately. The following formula expresses this property and can be checked in a 2-process system by Theorem 3.
  $(\forall i :: \ \mathsf{AG}((\hat{a}_i \ \Rightarrow \ \mathsf{X}_s^{a_i}(\neg\hat{a}_i \ \mathsf{U} \ \hat{b}_i)) \ \wedge \ (\hat{b}_i \Rightarrow \mathsf{X}_s^{b_i}(\neg\hat{b}_i \ \mathsf{U} \ \hat{a}_i))))$

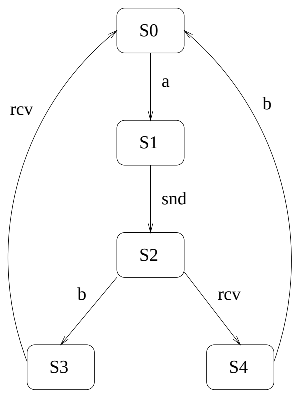

Fig. 3. Template process for Milner's Scheduler Protocol

6 Undecidability Results

Apt and Kozen [AK 86] show that determining if a temporal property holds for all instances of a parameterized network is undecidable, by a reduction from the non-halting problem for Turing machines. The essential idea is to have a ring of size n simulate the computation of a Turing machine for n steps. Suzuki [Su 88] proved a sharper result by showing that the undecidability holds even when the individual programs are finite-state and the ring is unidirectional.

We give a simpler proof of this result that also delineates sharply the boundary between decidable and undecidable token-ring systems. The proof is by a reduction from the non-halting problem for 2-counter machines. We show that undecidability arises even if the token takes values from a binary domain. The decidability results in this chapter hold for a token with a single value. Thus the information carrying capacity of the token defines the boundary between decidability and undecidability.

A 2-counter machine [Min 62] has four types of instructions: an increment and decrement for each counter, a zero-test, and a halt instruction. The halting problem for 2-counter machines is known to be undecidable. We reduce this problem to the parameterized model checking problem by using a ring of size n to simulate n steps of a 2-counter machine. Without loss of generality, we may suppose that the 2-counter machine ignores its input tape, and initially has both counter values equal to zero.

Each counter is implemented in unary by a single bit in each node of the ring. The bit takes values $\{up, down\}$ with the number of *up*'s in the global state giving the value of the counter. Each node runs a copy of the same program. Initially a single node is chosen nondeterministically, by giving it the token. That node executes the program of the 2-counter machine. A counter increment is done by circulating a token with an increment instruction. The node with a *down* bit for that counter that first receives this token changes its bit to *up*, then forwards the token with a "neutral" value. The decrement of a counter value occurs in a similar way, by changing the first *up* to *down*. The test for zero is done by circulating the token with value "zero"; if it is received by a process which has the corresponding bit set to *up*, then the token value is changed to "non-zero". If the increment token returns without the neutral value, this implies that the value of that counter is n, so the process simulating the 2-counter machine enters a "dead" state, which has no outgoing transitions.

This simple simulation uses seven values for the token: increment and decrement instructions for each counter, zero/non-zero values, and a neutral value. We can reduce is to a binary token by encoding each value in unary. A process passes one of these values to its neighbor by sending its unary code to the neighbor as a sequence of tokens with value 1, terminated by a token with value 0. The neighbor accumulates the unary count on receiving 1-tokens, but retransmits them with value 0, which return to the sender, without effecting any change on the local states of the processes in between.

Consider the property $(\forall i :: \mathit{initially\_token}_i \Rightarrow \mathsf{AG}\neg \mathit{halt}_i)$, which expresses the condition that the node executing the 2-counter machine program does not reach a halt state. If the 2-counter machine halts, then it must do so in n steps, for some n, so by the simulation, this property is false for a ring of size at least n. If it does not halt, then this property is true for rings of all sizes. Thus, the property is true for all instances iff the machine does not halt. Hence, parameterized verification is undecidable, specifically *co-RE*.

7 Related Work and Conclusions

Among related work, [AK 86,Su 88] show that the problem of automatically checking a specification for every instance of a parameterized system is in general undecidable. Positive results include those of Clarke, Grumberg and Browne [CG 87,BCG 89]; however, their method requires the manual construction of bisimulations or that of a *closure process* which represents computations of an arbitrary number of processes. [KM 89] and [WL 89] introduce the related notion of a *process invariant*. All these methods rely on human ingenuity to manually construct a suitable process closure or invariant. [GS 92] use automata-theoretic methods to construct process closures for processes connected in a complete network, and use them to establish single index properties. Multi-index properties can be indirectly catered for, but the complexity then becomes multi-exponential. In any case, this does not provide an algorithm for ring networks. Vernier [V 93] has developed a model-checking algorithm for a class of parameterized systems, however, the complexity is high.

Also related are the results of Shtadler and Grumberg [SG 89], who use a network grammar to specify a communication topology, and those of [LSY 94] which deal with ring networks of Petri nets. [SG 89] show that if certain sufficient conditions are satisfied, then every network generated by the grammar satisfies specifications written in linear-time temporal logic. The sufficiency check, however, may require time exponential in the size of an individual process. [LSY 94] contains another (exponential-time) sufficiency test which is used to show that certain parameterized protocols on rings satisfy a single-index linear-time specification. The work in [SG 89] has been extended to global safety properties in [CJ 95,CGJ 95], which presents a construction of a invariant process where the states of the process are regular expressions denoting global states of arbitrarily large instances. Sistla [Si 97] presents a method for generating an automaton over 2-dimensional strings that forms an invariant process for processes in a linear or circular array; however, it is not guaranteed that the method terminates in general. Our results for token rings are more general than those that may be derived from these earlier results, as they allow multi-index, branching-time properties to be checked with polynomial cost.

In this paper, we have shown that in our ring framework we can reduce reasoning about an n-ring of homogeneous processes, for arbitrarily large consisderation of a small ring of size c. The cutoff c depends on the indices of the propeerty being checked but is typically quite small, in the range of 2 to 5. each process in the ring, provided that it follows the token-passing discipline. The ring of size equal to the cutoff is analogous to a closure process, but is trivial to construct. In addition, the result holds even if the transition graph of the template process is not finite, provided that it is finite-branching. If it is finite, then an automated tool such as SMV [McM92], or the Concurrency Workbench [CPS 89] can be used to model check the desired property for the small ring. This check can be done in time polynomial in the size of a process. A preliminary version of these results was reported in [EN 95]. This paper improves upon those results in two ways. The more general indexed $\mathsf{CTL}^*\backslash\mathsf{X}$ is catered for here, instead of the flat logic amounting to $\mathsf{LTL}\backslash\mathsf{X}$ there. Moreover, the proofs have been simplified.

We believe the cutoff technique is a useful one. More recently, Emerson and Kahlon [EK 00] have established cutoff results for a different model of computation based on synchronization skeletons. Another interesting problem to consider is whether there are conditions under which a similar results hold for ring systems with multiple-valued tokens. The simplicity of the 2-counter machine simulation suggests that such conditions should be semantic instead of syntactic, perhaps in the form of invariance constraints on the communication patterns.

References

[AK 86] Apt, K., Kozen, D. Limits for automatic verification of finite-state concurrent systems. IPL 15, pp. 307-309.

[BCG 88] Browne, M. C., Clarke, E. M., Grumberg, O. Characterizing Finite Kripke Structures in Propositional Temporal Logic, Theor. Comp. Sci., vol. 59, pp. 115–131, 1988.

[BCG 89] Browne, M. C., Clarke, E. M., Grumberg, O. Reasoning about Networks with Many Identical Finite State Processes, Information and Computation, vol. 81, no. 1, pp. 13–31, April 1989.

[CE 81] Clarke, E. M., Emerson, E. A. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic, in *Workshop on Logics of Programs*, Springer-Verlag LNCS 131.

[CES 86] Clarke, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite-State Concurrent Systems using Temporal Logic, *ACM Trans. Prog Lang and Sys.*, vol. 8, no. 2, pp. 244-263, April 1986.

[CFJ 93] Clarke, E. M., Filkorn, T., Jha, S. Exploiting Symmetry in Temporal Logic Model Checking, 5th CAV, Springer-Verlag LNCS 697.

[CG 87] Clarke, E. M., Grumberg, O. Avoiding the State Explosion Problem in Temporal Logic Model Checking Algorithms, PODC 1987.

[CGJ 95] Clarke, E.M., Grumberg, O., Jha, S. Verifying Parameterized Networks using Abstraction and Regular Languages. CONCUR 95. Full version in *ACM TOPLAS*, 1997.

[CJ 95] Clarke, E.M., Jha, S. Symmetry and Induction in Model Checking. Computer Science Today 1995, pp. 455-470

[CPS 89] Cleaveland, R., Parrow, J., Steffen, B. The Concurrency Workbench. In J.Sifakis (ed), *Automatic Verification Methods for Finite State Systems*, Springer-Verlag, LNCS 407.

[D 85] Dijkstra, E. W. Invariance and non-determinacy, in *Mathematical Logic and Programming Languages*, Prentice-Hall International Series in Computer Science. eds. C.A.R Hoare and J.C Shepherdson.

[DS 90] Dijkstra, E. W., Scholten. C. S. *Predicate Calculus and Program Semantics*, Springer-Verlag, 1990.

[Em 90] Emerson, E. A. Temporal and Modal Logic, in Handbook of Theoretical Computer Science, (J. van Leeuwen, ed.), Elsevier/North-Holland, 1991.

[EH 86] Emerson, E.A., Halpern, J. Y. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic Journal of the ACM, Vol. 33, pp. 151-178, 1986.

[EK 00] E. Allen Emerson, Vineet Kahlon, " Reducing Model Checking of the Many to the Few", Conf. on Automated Deduction (CADE) 2000, pp. 236-254

[EN 95] Emerson, E.A., Namjoshi, K.S. Reasoning about Rings. *Proc. ACM Symposium on Principles of Programming Languages*, 1995.

[ES 93] Emerson, E. A., Sistla, A. P. Symmetry and Model Checking, 5th CAV, Springer-Verlag LNCS 697.

[GS 92] German, S. M., Sistla, A. P. Reasoning about Systems with Many Processes. J.ACM, Vol. 39, Number 3, July 1992.

[Kel 76] Keller, R. M. Formal verification of parallel programs. CACM, July 1976.

[KM 89] Kurshan, R. P., McMillan, K. A Structural Induction Theorem for Processes, PODC 1989.

[LSY 94] Li, J., Suzuki, I., Yamashita, M. A New Structural Induction Theorem for Rings of Temporal Petri Nets. IEEE Trans. Soft. Engg, vol. 20, No. 2, February 1994.

[Mil 90] Milner, R. *Communication and Concurrency*, Prentice-Hall International Series in Computer Science. ed. C.A.R Hoare.

[Min 62] Minsky, M. Computation : Finite and Infinite Machines, Prentice-Hall, 1962.

[McM92] McMillan, K., Symbolic Model Checking: An Approach to the State Explosion Problem, Ph.D. Thesis, Carnegie-Mellon University, 1992.

[MC 88] Misra, J., Chandy, K. M. *Parallel Program Design : A Foundation*, Addison-Wesley Publishers, 1988.

[MP 94] Manna, Z., Pnueli, A. Verification of Parameterized Programs. in *Specification and Validation Methods* (E. Borger, ed.), Oxford University Press, pp. 167-230, 1994.

[Pn 77] Pnueli, A. The Temporal Logic of Programs. FOCS 1977.

[QS 82] Queille, J.P., Sifakis, J. Specification and verification of concurrent systems in CESAR, in Proc. of the 5th International Symposium on Programming, LNCS 137, 1982.

[SG 89] Shtadler, Z., Grumberg, O. Network Grammars, Communication Behaviours and Automatic Verification. In J.Sifakis (ed), *Automatic Verification Methods for Finite State Systems*, Springer-Verlag, LNCS 407.

[Si 97] Sistla, A.P. Parametrized Verification of Linear Networks Using Automata as Invariants. CAV 1997, pp. 412-423

[Su 88] Suzuki, I. Proving properties of a ring of finite state machines. IPL 28, pp. 213-214.

[VW 86] Vardi, M., Wolper, P. An Automata-theoretic Approach to Automatic Program Verification, Proc. IEEE LICS, pp. 332-344, 1986.

[V 93] Vernier, I. Specification and Verification of Parameterized Parallel Programs. Proc. 8th International Symposium on Computer and Information Sciences, Istanbul, Turkey, pp. 622-625.

[WL 89] Wolper, P., Lovinfosse, V. 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.

A Appendix

A.1 Proof of Theorem 2

Theorem 2 (Reduction Theorem) *Let $I \subseteq [n]$ and $J \subseteq [k]$ be sets of indices. Let $h : I \rightarrow J$ be a bijection such that for any $i, j \in I$,*

1. *$i \leq j$ iff $h(i) \leq h(j)$, and*
2. *$(i : j)_n \neq \emptyset$ iff $(h(i) : h(j))_k \neq \emptyset$.*

Then $M_n|_I \sim M_k|_J$.

Proof Sketch.

We define first a relation R_{ij} between states of M_i and M_j w.r.t. a bijection $f : X \rightarrow Y$ that satisfies conditions analogous to those satisfied by the function h above. We then instantiate i, j and f in various ways to form a relation between $M_n|_I$ and $M_k|_J$. In the following, let $tok_l.s$ be the index of the process in M_l that possesses the token in state s.

From f, one can derive the renaming function f_{ij}^A for indexed actions and the renaming function f_{ij}^P for indexed propositions as follows: $f_{ij}^A(a_l) = a_{f(l)}$ and $f_{ij}^P(P_l) = P_{f(l)}$. Let R_{ij} be the relation between states of $M_i|_X$ and $M_j|_Y$, defined w.r.t. f as follows.

$s\ R_{ij}\ t$ iff

1. The local states of D_i in s and D_j in t are identical (recall that D is the initial token distribution process),
2. The state of the processes indexed by X in s is identical to the state of the processes indexed by Y in t, up to f: $f_{ij}^P(\lambda(s)|_X) = \lambda(t)|_Y$,
3. The token is at a process l in s iff it is at the corresponding process $f(l)$ in t: $(\forall l : l \in X : tok_i(s) = l \equiv tok_j(t) = f(l))$, and
4. The token is between processes l and m in s iff it is between the corresponding processes in t: $(\forall l, m : l, m \in X : tok_i(s) \in (l : m)_i \ \equiv\ tok_j(t) \in (f(l) : f(m))_j)$.

We instantiate this definition in the following ways. Let B_0 be the union of the relations R_{nn} and R_{kk} obtained by taking f to be the identity on I and J respectively. Let B_1 be the relation R_{nk} obtained by instantiating f with the function h in the hypothesis. Let B_2 be the relation R_{kn} obtained by instantiating f with the function h^{-1}. Finally, let $B = B_0 \ \cup\ B_1 \ \cup\ B_2$. It is straightforward to check that B_0 is an equivalence relation and that $B_0; B_i; B_0 \subseteq B_i$, for all i.

Suppose sB_1t. Then s is a state in M_n and t is a state in M_k, and by definition, $h_1^P(\lambda(s)|_I) = \lambda(t)|_J$.

1. s is the initial state of M_n.

The only enabled action is the token transfer to a process, say process i. As sB_1t, t is the initial state of M_k, hence the token transfer action is enabled at t. If $i \in I$, let the transfer from t be to process $h(i)$. Otherwise, there is a pair k, l of indices in I such that $i \in (k : l)$, and $(k : l)$ does not contain an index in I. As sBt, $(h(k) : h(l)) \neq \emptyset$, so let the transfer from t be to some process with index in $(h(k) : h(l))$. It is straightforward to check that the resulting states are related. These single step transitions are the only sequences in $Fin_B(s)$.

2. s is not the initial state of M_n.

(a) Let p be a path in $Fin_B(s)$. Since the only non-τ actions in $M_k|_I$ are those of processes in I, the last action in p is a move by some process i in I. This action can be either a local move, or a token transfer through a send or receive action.

– The action is an internal move by process i. As process $h(i)$ has the same local state, the internal move can be performed at that process, and the resulting states are related by B_1.
– The action is a token send by process i. Note that the states of process i and $h(i)$ are identical. Hence, the same token send can be performed at $h(i)$, leading to a state related by B_1 to the one resulting from the process i action.
– The action is a receipt of the token by process i from process $i-1$, for $i \in I$ (arithmetic is modulo n). Hence, the token is in the range $(k:i)$ for some (least) k in I. As s and t are related by B_1, the token in in the range $(h(k):h(i))$ in t. If process $h(i)-1$ is ready to send the token, the receive action may be executed at process $h(i)$, leading to a related state.
  Otherwise, the token is at some process j in $(h(k):h(i))$. By the structure of the processes (cf. Section 3), there is a path where the process j eventually reaches a blocking send action and processes without the token reach a blocking receive action. At the end of this path, the actions that transfer the token from process j to process $j+1$ are enabled. Iteratively appending such paths for all processes l from j to $h(i)-1$, one obtains a path where the token moves from process j to a state in process $h(i)-1$ where the token transfer to $h(i)$ is enabled, by a sequence of unobservable actions in $M_k|_J$ and through a sequence of states that are all related by B_0 to the state t. By the earlier argument, the token transfer to $h(i)$ can now be executed, leading to a state related by B_1 to the state resulting from the token transfer to process i in M_n.

(b) The other case, that of infinite sequences from s that are in $Inf_B(s)$, does not arise because every process alternates between send and receive actions. This induces the token to move from one process to the next. Thus, the token eventually either reaches or leaves a process in I, which induces a change to a state not equivalent to s.

The proof above is for the case where s is a state in M_n and t a state in M_k. A similar proof shows the other cases. It follows that B is a block bisimulation. Hence, $M_n|_I \sim M_k|_J$. Notice that this proof shows also that the non-τ actions are matched by B up to h^A. □

A.2 Block Bisimulation

We show below that Stuttering Bisimulation and Block Bisimulation coincide for total Kripke Structures. We also present a proof for Theorem 1.

Let A be a Kripke Structure, $A = (Q, \{\tau\}, \delta, \lambda, L, I)$. [BCG 88] define stuttering bisimulation over finite state Kripke Structures with a total transition relation.

Definition 5 (Stuttering Bisimulation) *A relation B on A is a stuttering bisimulation iff B is symmetric, and for any s, t such that sBt,*

1. *$\lambda(s) = \lambda(t)$, and*
2. *For every fullpath p starting at s, there is a fullpath q starting at t such that p matches q by B.*

where a *partition* of a fullpath $p = (\sigma, \alpha)$ is a division of p into *segments*, where each segment is a sub-path of the form $(\sigma_i \ldots \sigma_{i+k}, \alpha_i \ldots \alpha_{i+k})$, for some non-negative i, k, and fullpaths p and q *match* w.r.t. B iff they can be partitioned into an equal number of non-empty, finite segments, such that for each segment k, every state in the kth segment of p is related by B to every state in the kth segment of q.

It can be shown, using the Knaster-Tarski theorem with the above definition (as in [Mil 90]), that the greatest stuttering bisimulation exists and is an equivalence relation. Two states are stuttering equivalent iff there is a stuttering bisimulation that includes the pair.

Theorem 3 *Let $B = \{B_0\}$ be a block bisimulation. Then B is also a stuttering bisimulation.*

Proof. B is an equivalence relation by (1) of Defn. 3. Let (s,t) be an arbitrary pair in B. As h_0 is the identity function, $\lambda(s) = \lambda(t)$ by 2(a). Consider an infinite fullpath p starting at s. If $p \in \mathit{Inf}_B(s)$, then by 2(c) of Defn. 3, there is a fullpath q starting at t such that $q \in \mathit{Inf}_B(t)$. So p and q may be partitioned into the same number of segments which consist of states related by B. If p is not in $\mathit{Inf}_B(s)$, then there is a state in a B-class different from s on the path. Hence, there is a maximal prefix p' such that $p' \in \mathit{Fin}_B(s)$. By 2(b), there is a finite path q' in $\mathit{Fin}_B(t)$ such that $\mathit{endS}(p')\ B\ \mathit{endS}(q')$. p' and q' without the end states form a pair of matching segments, and $(\mathit{endS}(p'), \mathit{endS}(q'))$ form the start of a new pair of corresponding segments. Continuing in this manner, one can inductively define a fullpath q starting at t, and partitions of p and q such that p and q match w.r.t. B. Hence, B is a stuttering bisimulation. $\square$

Theorem 4 *Let B_0 be the greatest stuttering bisimulation. Then $B = \{B_0\}$ is a block bisimulation.*

Proof. The greatest stuttering bisimulation is an equivalence relation. Let (s,t) be an arbitrary pair in B_0. Then, $\lambda(s) = \lambda(t)$, hence, condition 2(a) is satisfied.

Consider a path p in $\mathit{Inf}_B(s)$. As p is a fullpath, by (2) of Defn. 5, there is a fullpath q starting at t such that p matches q w.r.t. B_0. This implies that q is in $\mathit{Inf}_B(t)$, so 2(c) holds. Consider a finite path p' in $\mathit{Fin}_B(s)$. Let γ be a path such that $p = p';\gamma$ is a fullpath starting at s. By condition 2(b) of Defn. 5, there is a matching fullpath q starting at t. Since p' ends with a state in a different B-class than s, by the definition of matching paths, there is an initial segment q' of q and the state v following q' such that $q';v$ is in $\mathit{Fin}_B(t)$ and $\mathit{endS}(p')\ B_0\ v$. Hence, condition 2(b) of the block bisimulation definition is satisfied. Thus, B is a block bisimulation. $\square$

A.3 Proof of Theorem 1

Theorem 1 *Let $B = \{B_i\}$ be a block bisimulation on LTS A. For $(s,t) \in B_i$ and any formula f of $\mathsf{CTL}^*_\Sigma\backslash\mathsf{X}$, $A, s \models f$ iff $A, t \models h_i(f)$.* $\square$

Proof.

The proof is by structural induction on formulas of $\mathsf{CTL}^*_\Sigma\backslash\mathsf{X}$.

Basis: $f \in AP$.

$$
\begin{array}{ll}
& M, s \models f \\
\Leftrightarrow & (\text{ definition }) \\
& f \in \lambda(s) \\
\Leftrightarrow & (\ h_i^{AP} \text{ is a bijection on } AP\ ) \\
& h_i(f) \in h_i(\lambda(s)) \\
\Leftrightarrow & (\text{ condition 2(a) of Defn. 3 }) \\
& h_i(f) \in \lambda(t) \\
\Leftrightarrow & (\text{ definition }) \\
& M, t \models h_i(f)
\end{array}
$$

Inductive case: If f has the form $g_0 \wedge g_1$ or $\neg g$, then the proof follows directly from the inductive hypothesis. Consider the case where f has the form $\mathsf{E}g$, for a path formula g.

$$
\begin{array}{ll}
& M, s \models \mathsf{E}g \\
\Leftrightarrow & (\text{ definition }) \\
& (\exists p : \mathit{fullpath}(p, s) : M, p \models g) \\
\Leftrightarrow & (\text{ by following proof }) \\
& (\exists q : \mathit{fullpath}(q, t) : M, q \models h_i(g))
\end{array}
$$

$\Leftrightarrow$ (definition)
$M, t \models \mathsf{E}h_i(g)$
$\Leftrightarrow$ (definition)
$M, t \models h_i(f)$

To justify the second step of the proof above, suppose that $p = (\sigma, \alpha)$ is a fullpath starting at s. As $(s,t) \in B_i$, by Defn. 3 and a simple inductive argument involving 2(b) and 2(c), there is a fullpath $q = (\delta, \beta)$ starting at t such that the state sequences of p and q match w.r.t. B_i, and the action sequences of p and q match up to h_i. By the inductive hypothesis, states related by B_i agree up to h_i on the truth value of sub-formulas of the path formula g that are state formulas. The inductive argument where g has the form $g_0 \wedge g_1$ or $\neg g_0$ is straightforward. For the case where g has the form $\mathsf{X}_s^a g$,

$M, p \models \mathsf{X}^a g$
$\Leftrightarrow$ (definition)
$(\exists i : \alpha[0 \ldots i-1] \in \tau^* \ \wedge \ \alpha_i = a : p^{i+1} \models g)$
$\Leftrightarrow$ (p and q match w.r.t. B_i; observation below, inductive hypothesis)
$(\exists k : \beta[0 \ldots k-1] \in \tau^* \ \wedge \ \beta_k = h_i^{\Sigma}(a) : p^{k+1} \models h_i(g))$
$\Leftrightarrow$ (definition)
$M, q \models X_s^{h_i(a)} h_i(g)$
$\Leftrightarrow$ (definition)
$M, q \models h_i(X_s^a g)$

Consider the case where g has the form $g_0 \mathsf{U} g_1$.

$M, p \models g_0 \mathsf{U} g_1$
$\Leftrightarrow$ (definition)
$(\exists i : M, p^i \models g_1 \ \wedge \ (\forall j : j < i : M, p^j \models g_0))$
$\Leftrightarrow$ (p and q match w.r.t. B_i; observation below, inductive hypothesis)
$(\exists k : M, q^k \models h_i(g_1) \ \wedge \ (\forall l : l < k : M, p^l \models h_i(g_0)))$
$\Leftrightarrow$ (definition)
$M, q \models h_i(g_0) \mathsf{U} h_i(g_1)$
$\Leftrightarrow$ (definition)
$M, q \models h_i(g_0 \mathsf{U} g_1)$

In both proofs, the second step follows from the following observation: for matching paths, every position on one path has a corresponding position (in the corresponding segment) from which the suffix paths also match w.r.t. B_i and h_i. $\square$