Reasoning about Rings*

E. Allen Emerson Kedar S. Namjoshi

Department of Computer Sciences
The University of Texas at Austin, U.S.A.

Abstract

The ring is a useful means of structuring concurrent processes. Processes communicate by passing a token in a fixed direction; the process that possesses the token is allowed to perfrom certain actions. Usually, correctness properties are expected to hold irrespective of the size of the ring. We show that the problem 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 one of the most useful ways for structuring systems 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 direct proof by hand [MC 88, MP 94], which requires considerable ingenuity, and can in practice be done only for reasonably simple protocols.

In this paper, we show that, for *any* system of many isomorphic processes organized in a ring which communicate through a token used as a signal, 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, CES 86] may be applied for the verification of the parameterized system.

The logic in which correctness properties are expressed is the branching time logic CTL^* without the *next-time* operator X, which we denote by $\mathsf{CTL}^*\backslash\mathsf{X}$ [BCG 89]. Such formulas are insensitive to "stuttering" (repeated occurences of the same state). Since the formulas are to hold of rings of various sizes, it seems reasonable to make them free of next-time requirements, which will in general vary among rings of various sizes. It is, however, possible to handle a "next-action by process i" modality.

Correctness properties of parameterized systems are typically expressed as properties in an indexed temporal logic. Let $g(i)$ be a formula in which every proposition is indexed by the variable i, and $g(i,j)$ be a formula in which every proposition is indexed either by the variable i or by the variable j. [1] We consider correctness properties of the form $\bigwedge_i g(i)$ (every process satisfies $g(i)$) and $\bigwedge_{i,j:i\neq j} g(i,j)$ (every distinct pair of processes satisfies $g(i,j)$). The range of i and j is $[0..n-1]$, where n is the size of the ring over which the formula is evaluated. This range is left implicit for conciseness. For instance, mutual exclusion can be expressed as $\bigwedge_{i,j:i\neq j} \mathsf{AG}\neg(\mathit{critical}_i \wedge \mathit{critical}_j)$, and eventual access by $\bigwedge_i \mathsf{AG}(\mathit{trying}_i \Rightarrow \mathit{critical}_i)$. The specific results are as follows :

*This work was supported in part by NSF grant CCR-9415496 and Semiconductor Research Corporation Contract 94-DP-388. The authors can be reached at `emerson, kedar@cs.utexas.edu`. A preliminary version of this paper was presented at the ACM Symposium on Principles of Programming Languages (POPL), 1995.

[1] Thus the instances $g(1)$ and $g(2)$ of the formula $g(i)$ are isomorphic up to re-indexing, and similarly for $g(i,j)$.

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

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 $\bigwedge_i g(i)$, which ranges over all possible process indices in an instance of the system, to that of one with a particular index, say $g(0)$. We then establish the existence of a correspondence 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 correspondence established is for the projection of the state transition graph on the particular process index; the symmetry then allows us to establish the result. The main observation of the correspondence proof is that a segment of the ring not containing any observable processes (e.g., for $\bigwedge_i g(i)$, one not containing process 0) has behaviour similar to that of a single process.

The rest of 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 that the parameterized verification task is undecidable if tokens can carry 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

Quantified expressions are written in the format $(\mathbf{Q}x : r : p)$, where $\mathbf{Q}$ is the quantifier (e.g., $\exists, \forall, min, max$), x the bound variable, r the range, and p the expression being quantified (cf. [DS 90]). 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 powerset of a set S is denoted by $\mathcal{P}(S)$. For a binary relation R, we often write $s\ R\ t$ instead of $(s, t) \in R$ for readability.

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

$$\begin{array}{ll} & P \\ op & (\text{ hint why } P\ op\ Q\ ) \\ & Q \end{array}$$

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 iff(if-and-only-if).

2.2 Labelled Transition Systems

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

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

- *Q is a nonempty set of states,*
- *Σ is a transition (or action) alphabet, possibly containing 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, with the condition that if $(s, \tau, t) \in \delta$ then $\lambda(s) = \lambda(t)$.*
- *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 $s \overset{A}{\to} t$ for $(\exists a : a \in \Sigma : s \xrightarrow{a}_A t)$. The subscript A is often elided when a specific structure is under consideration. A *path* σ of A is a (finite or infinite) sequence of states such that for every i such that $i + 1 < \mathit{length}(\sigma)$, $\sigma_i \overset{A}{\to} \sigma_{i+1}$. The length of a path σ is denoted by $\mathit{length}(\sigma)$, and is the number of states on σ. A *fullpath* is a maximal path; i.e., either it is infinite or the last state has no successor. A *computation* is a path that starts at an initial state of A.

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 defined by a structure $(Q, \delta, \lambda, AP, I)$, where*

- *Q is a nonempty set of states,*
- *$\delta \subseteq Q \times Q$ is the transition relation,*
- *$\lambda : Q \to \mathcal{P}(AP)$ is a labelling function on the states,*
- *AP is a nonempty set of* atomic *propositions,*
- *$I \subseteq Q$ is the set of initial states.*

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 $B = (Q_B, \Sigma_B, \delta_B, \lambda_B, L_B, I_B)$, such that $L_A \cap L_B = \emptyset$, is defined with respect to a *synchronizer* structure $\Gamma = (\Gamma_A, \Gamma_B, \bar{\ })$, where $\Gamma_A \subseteq \Sigma_A \backslash \{\tau\}$, $\Gamma_B \subseteq \Sigma_B \backslash \{\tau\}$, and $\bar{\ } : \Gamma_A \to \Gamma_B$ is a bijection. Informally, the sets in the synchronizer represent the actions that the processes synchronize on. 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. $\Sigma_{AB} = (\Sigma_A \cup \Sigma_B) \backslash (\Gamma_A \cup \Gamma_B)$,
3. δ_{AB} is defined by :
   $(s, t) \xrightarrow{a}_{AB} (u, v)$ iff
   (a) $a \notin \Gamma_A \wedge s \xrightarrow{a}_A u \wedge t = v$ (a move by A only), or
   (b) $a \notin \Gamma_B \wedge t \xrightarrow{a}_B v \wedge s = u$ (a move by B only), or
   (c) $a = \tau \wedge (\exists b :: s \xrightarrow{b}_A u \wedge t \xrightarrow{\bar{b}}_B v)$ (a synchronized move)
4. $\lambda_{AB}(s, t) = \lambda_A(s) \cup \lambda_B(t)$,
5. $L_{AB} = L_A \cup L_B$, and
6. $q^0_{AB} = I_A \times I_B$.

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

2.4 LTS Projection

For two LTS's $A = (Q_A, \Sigma_A, \delta_A, \lambda_A, L_A, I_A)$ and $B = (Q_B, \Sigma_B, \delta_B, \lambda_B, L_B, I_B)$, composed to form $C = A \parallel_\Gamma B$, the projection of C on A is defined as the LTS $C|_A$ given by $C|_A = (Q, \Sigma, \delta, \lambda, L, I)$, 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) $a = \tau \wedge (\exists b : b \in \Sigma_B : s \xrightarrow{b}_C t)$
4. $\lambda(s) = \lambda_C(s) \cap L_A$,
5. $L = L_A$, and
6. $I = I_C$.

The effect of projecting on to A is just to elide the labels on actions performed by the B-component, replacing them with the silent action τ.

2.5 Temporal Logics and their Semantics

We will define the syntax and semantics of the *Full Branching Time Logic* CTL^*. The other logics considered (CTL and LTL) are sublogics of CTL^*, and will be defined as such.

CTL^* (read as "Computation Tree Logic - star") [EH 82] 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 nexttime", and "weak until", respectively), and they are allowed to appear in virtually arbitrary combinations. 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 :

(S1) Any atomic proposition P is a state formula.

(S2) If p, q are state formulas, then so are $p \wedge q$, $\neg p$.

(S3) If p is a path formula then $\mathsf{E}p$ is a state formula.

(P1) Any state formula p is also a path formula.

(P2) If p, q are path formulas, then so are $p \wedge q$, $\neg p$.

(P3) If p, q are path formulas then so are $\mathsf{X}_s p$ and $p \mathsf{U}_w q$.

The semantics of a formula is defined with respect to a Kripke Structure $M = (S, R, L, AP, I)$. We write $M, s \models p$ ($M, x \models p$) to mean that state formula p (path formula p) is true in structure M at state s (of fullpath x, resp.). When M is understood, we write simply $s \models p$ ($x \models p$). We define $\models$ inductively using the convention that x denotes a fullpath and x^i denotes the suffix fullpath starting at the ith position in x, provided $i < \mathit{length}(x)$. For a state s :

(S1) $s \models P$ iff $P \in L(s)$ for atomic proposition P

(S2) $s \models p \wedge q$ iff $s \models p$ and $s \models q$,
$s \models \neg p$ iff not ($s \models p$)

(S3) $s \models \mathsf{E}p$ iff for some fullpath x starting at s, $x \models p$

For a fullpath x :

(P1) $x \models p$ iff $x_0 \models p$, for any state formula p

(P2) $x \models p \wedge q$ iff $x \models p$ and $x \models q$,
$x \models \neg p$ iff not $(x \models p)$

(P3) $x \models \mathsf{X}_s p$ iff x^1 is defined and $x^1 \models p$,
$x \models (p \,\mathsf{U}_w\, q)$ iff for all $i < |x|$, if for all $j \leq i$ $x^j \models \neg q$ then $x^i \models p$

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

Other connectives can then be defined as abbreviations in the usual way: $p \vee q \equiv \neg(\neg p \wedge \neg q)$, $p \Rightarrow q \equiv \neg p \vee q$, $p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$, $\mathsf{A}p \equiv \neg\,\mathsf{E}\,\neg\, p$, $\mathsf{G}p \equiv p \,\mathsf{U}_w\, false$, and $\mathsf{F}p \equiv \neg\,\mathsf{G}\,\neg\, p$. Further operators may also be defined as follows:
$\mathsf{X}_w p \equiv \neg \mathsf{X}_s \neg p$ is the weak nexttime,
$p \mathsf{U}_s q \equiv (p \mathsf{U}_w q) \wedge \mathsf{F}q$ is the strong until,
$\overset{\infty}{\mathsf{F}} p \equiv \mathsf{GFX}_s p$ means infinitely often p,
$\overset{\infty}{\mathsf{G}} p \equiv \mathsf{FGX}_s p$ means almost everywhere p,
inf $\equiv \mathsf{GX}_s true$ means the path is infinite, and
fin $\equiv \mathsf{FX}_w false$ means the path is finite.

Sublogics of CTL^* 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, $M, x \models f$.

ACTL^* is obtained by considering only positive formulas, built up from the literals (propositions or negated propositions), without negation, and using only the A path quantification operator. The dual logic is known as ECTL^*.

$\mathsf{CTL}^* \backslash \mathsf{X}$ is obtained by leaving out the operator X_s. This makes the logic insensitive to *stuttering* (repetitions of the same state).

CTL (Computation Tree Logic) is obtained by restricting the kinds of path formulas possible. The new rules are :

(S1) Any atomic proposition P is a state formula.

(S2) If p, q are state formulas, then so are $p \wedge q$, $\neg p$.

(S3) If p is a path formula then $\mathsf{E}p$ is a state formula.

(P1) Any state formula p is also a path formula.

(P3) If p, q are path formulas then so are $\mathsf{X}_s p$ and $p \mathsf{U}_w q$.

The rules ensure that path quantifiers are always attached to a single temporal operator. Thus, $\mathsf{AG}(p \Rightarrow \mathsf{AF}q)$ is a CTL formula, while $\mathsf{AG}(p \Rightarrow \mathsf{F}q)$ is not.

CTL^* itself is a sublogic of the mu-calculus, which is defined as the least set of formulas that satisfy :

- Any atomic proposition P, and any variable symbol Z is a formula.

- If p, q are formulas, then so are $p \wedge q$, $\neg p$.
- If p is a formula, then $\mathsf{EX}p$ is a formula.
- If p is a formula, then $\mu Z\ p$ is a formula.

The paper by Emerson [Em 90] contains a detailed survey of these logics and a comparison of their expressive powers.

2.6 Block Bisimulation

Let A be an LTS, $A = (Q, \Sigma, \delta, \lambda, L, I)$. Let $\approx$ be an equivalence relation on Q. $MAXF_{\approx}(s)$ is the set of finite paths of the form $\sigma; v$, where σ begins at s ($\sigma_0 = s$), every state in σ is in the same $\approx$-class as s $((\forall i : i < \mathit{length}.\sigma : \sigma_i \approx s))$, and v is in a different class $\neg(v \approx s)$. $MAXI_{\approx}(s)$ is the set of all fullpaths σ, where σ begins at s and every state in σ is in the same $\approx$-class as s.

Definition 3 (Block Bisimulation) *Let $\approx$ be an equivalence relation on Q, and let h be a bijection on AP. A relation B on Q is a block bisimulation on A w.r.t. $(\approx, h)$ iff*

1. *B is symmetric,*
2. *B is monotonic w.r.t. $\approx$: For any s, t, u, v if $(s, t) \in B$, $s \approx u$, and $t \approx v$, then $(u, v) \in B$.*
3. *Whenever $(s, t) \in B$ then*
   (a) $h(\lambda(s)) = \lambda(t)$,
   (b) $(\forall \sigma, v : \sigma; v \in MAXF_{\approx}(s) : (\exists \delta, w : \delta; w \in MAXF_{\approx}(t) : vBw))$
   (c) $(\forall \sigma : \sigma \in MAXI_{\approx}(s) : (\exists \delta : \delta \in MAXI_{\approx}(t) : \mathit{length}(\delta) = \omega \equiv \mathit{length}(\sigma) = \omega))$

□

The term "block" bisimulation is used as the definition essentially compares paths consisting of states in the same $\approx$-class (block). In order to compare two LTS's A and B, form the disjoint union C of A and B and define a bisimulation on C. For a formula f, let $h(f)$ be the formula obtained by replacing each proposition P of f by $h(P)$. The following theorem is shown in the Appendix.

Theorem 1 *Let B be a block bisimulation on LTS A w.r.t. $(\approx, h)$. If $(s, t) \in B$, then for any formula f of* $\mathsf{CTL}^*\backslash\mathsf{X}$, *$A, s \models f$ iff $A, t \models h(f)$.* □

We say that states s and t are block equivalent, if there is some block bisimulation on A that includes the pair (s, t). We write $A \sim B$, for LTS's A and B iff every initial state in A has a block bisimilar 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 structures. 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, $Aut\ f = \{\pi \in Sym\ I : \pi(f) \equiv f\}$. $Auto\ f = Aut\ g_0 \cap Aut\ g_1 \ldots \cap Aut\ g_k$, where $g_0, g_1, \ldots g_k$ are the maximal propositional sub-formulas of f. For an LTG 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
auts, for a state s to be the set of permutations that when applied to s, map it to itself.

3 System Model

Informally, the token passing model is defined by the following :

- Initially, a nondeterministically chosen process has the unique token.
- A process has two types of transitions : those that are enabled only if the process has the token, and others which can be enabled without the process possessing the token. We let the system evolve according to pure nondeterminism, i.e., no fairness is assumed.
- The process with the token must eventually transmit it in a clockwise direction.

Formally, individual processes of a ring are constructed from a template process T, which is an LTS defined by :

1. The set of states, $Q \times \mathbf{B}$. The boolean component indicates possession of the token.
2. The set of actions, Σ, which is partitioned into Σ_f, the set of "free" actions, Σ_d, the set of "token dependent" actions, and $\{snd, rcv\}$, the set of token transfer actions.
3. The transition relation δ, where
   - For every $(q, b) \xrightarrow{a} (q', b') \in \delta$,
     (a) $a \in \Sigma_f \Rightarrow b \equiv b'$ (A free transition cannot change possession.)
     (b) $a \in \Sigma_d \Rightarrow b \wedge b'$ (A token dependent transition can execute only if the process possesses the token.)
     (c) $a = rcv \Rightarrow \neg b \ \wedge \ b'$ (A receive establishes possession of the token.)
     (d) $a = snd \Rightarrow b \wedge \neg b'$ (A send revokes possession of the token.)
   - For every (q, b) such that $(q^0, false) \xrightarrow{a} (q, b) \in \delta$, $a = rcv$. (The only possible initial action is a receive.)
   - rcv and snd actions alternate along every path in T.
   - δ is *total* in the first component. (The process is nonterminating)
4. The set of propositions is $Q \times \mathbf{B}$, and the labelling function is the identity function.
5. The initial state is $(q^0, false)$.

For a finite-state template process, it is possible to check the conditions on δ automatically, by model-checking a certain CTL formula.

An individual process K_i $(i \in [n], n \geq 1)$ in an instance of the system with n processes is defined by $K_i = \tau_i(T)$, where τ_i is the re-labelling defined by the action renamings $\{rcv_i/rcv, snd_{i+1}/snd\} \cup \{a_i/a : a \in \Sigma_f \cup \Sigma_d\}$ and the state renamings s_i/s for every state s. The instance 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 with a nondeterministically 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. Precisely, $[i : j]_n = \{i + k \mid k \in [n] \wedge (\forall l : l < k : i + l \neq j)\}$. Addition is modulo 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 from the definitions, $(i : i)_n$, $[i : i)_n$, and $(i : i]_n$ are all empty.* □

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 $\bigwedge_i g(i)$, $\bigwedge_{i,j:i\neq j} g(i,j)$, and $\bigwedge_{i,j:i\neq j} g(i,i+1,j)$. In each case, we first show by symmetry arguments, that in each instance of size n, it is sufficient to instantiate the quantifier $\bigwedge_i$ with some process index, say 0. We then prove that a system of size greater than the cutoff size satisfies this smaller formula iff the system of cutoff size does. 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 such that there is a bijection $h : I \to J$ 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 $M(n)|_I$ is block equivalent to $M(k)|_J$.

Proof. Let h^A be the derived renaming function for indexed actions from $M(n)$ defined by $h^A(a_i) = a_{h(i)}$. Let h^P be the derived renaming function for indexed propositions from $M(n)$ defined by $h^P(P_i) = P_{h(i)}$. We define a relation that is a block bisimulation on $M(n)$ and $M(k)$ w.r.t. h^P.

For a state s of $M(i)$, let $tok_i(s)$ be the index of the process that possesses the token in state s. Let R_{nk} be the relation between states of $M(n)$ and $M(k)$, defined as follows:

$s\ R_{nk}\ t$ iff

1. The local states of D_n in s and D_k in t are identical (recall that D_n is the initial token distribution process),
2. $h^P(\lambda(s)|_I) = \lambda(t)|_J$,
3. $(\forall i : i \in I : tok_n.s = i \equiv tok_k.t = h.i)$, and
4. $(\forall i, j : i, j \in I : tok_n(s) \in (i : j)_n \equiv tok_k(t) \in (h(i) : h(j))_k)$.

The relation R_{nn} is similar but for (2), which is modified to $\lambda(s)|_I = \lambda(t)_I$. Define the relation $\approx$ to be $R_{nn} \cup R_{kk}$. Let $B = R_{nk} \cup R_{kn}$. B is symmetric. It is straightforward to check that $\approx$ is an equivalence relation, and that B is monotonic w.r.t. $\approx$. The Appendix contains the proof that B is a block bisimulation w.r.t. $(\approx, h^P)$.

Informally, the bisimulation treats states as equivalent if corresponding processes (w.r.t. h) in I and J have the same local state (conditions 1,2), the token is at some process in I iff it is at the corresponding process in J (condition 3), and the token is in between two processes in I iff it is in between the corresponding processes in J (condition 4). $\square$

4.1 Properties of the form $\bigwedge_i g(i)$

Properties of the form $\bigwedge_i g(i)$ 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 ($\bigwedge_i \mathsf{AG}(trying_i \Rightarrow critical_i)$). 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, $A, q^0 \models \bigwedge_i g(i)$ iff $A, q^0 \models g(0)$.*

Proof.

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

$(LHS \Leftarrow RHS)$

$A, q^0 \models g(0)$
$\Rightarrow$ (π is a permutation)
$(\forall \pi:\ \pi \in Aut\ A:\ \pi(A), \pi(q^0) \models g(\pi(0)))$
iff ($\pi(A) = A$ as $\pi \in Aut\ A$, $\pi(q^0) = q^0$, as q^0 is symmetric)
$(\forall \pi:\ \pi \in Aut\ A:\ A, q^0 \models g(\pi(0)))$
$\Rightarrow$ (as $\mathcal{C}_n \leq Aut\ A$)
$(\forall \pi:\ \pi \in \mathcal{C}_n:\ A, q^0 \models g(\pi(0)))$
$\Rightarrow$ (The cyclic shifts in $\mathcal{C}_n$ drive 0 to i for every $i \in [n]$)
$(\forall i \in [n] : A, q^0 \models g(i))$
iff (by definition)
$A, q^0 \models \bigwedge_i g(i)$
□

Theorem 3 *Let f be a formula of the form $\bigwedge_i g(i)$, where $g(i)$ is a $\mathsf{CTL}^*\backslash\mathsf{X}$ 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 $n \geq 2$,
$M(n), q_n^0 \models f$
iff (the initial state is symmetric, lemma 1)
$M(n), q_n^0 \models g(0)$
iff (as $g(0)$ refers only to propositions indexed by 0)
$M(n)|_{\{0\}}, q_n^0 \models g(0)$
iff (By Theorem 2, $M(n)|_{\{0\}}$ and $M(2)|_{\{0\}}$ are block equivalent)
$M(2)|_{\{0\}}, q_2^0 \models g(0)$
iff (as $g(0)$ refers only to propositions indexed by 0)
$M(2), q_2^0 \models g(0)$
iff (by lemma 1)
$M(2), q_2^0 \models f$
□

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

Formulas with the form $\bigwedge_i g(i, i+1)$ can express properties of neighboring pairs of processes. For instance, the exclusion property for the Dining Philosophers problem may be expressed as $\bigwedge_i \mathsf{AG}(\neg(eating_i \wedge eating_{i+1}))$. 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 $A, q^0 \models \bigwedge_i g(i, i+1)$ 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 $\bigwedge_i g(i, i+1)$, where $g(i, i+1)$ is a $\mathsf{CTL}^*\backslash\mathsf{X}$ formula. Then, $(\forall n : n \geq 3 : M(n), q_n^0 \models f\ \equiv\ M(3), q_3^0 \models f)$* □

4.3 Properties of the form $\bigwedge_{i,j:i\neq j} g(i,j)$

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

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,*
$A, q^0 \models \bigwedge_{i,j:i\neq j} g(i,j)$ *iff* $A, q^0 \models \bigwedge_{j:j\neq 0} g(0,j)$.

Proof.

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

$(LHS \Leftarrow RHS)$

$A, q^0 \models \bigwedge_{j:j\neq 0} g(0,j)$
$\Rightarrow$ (π is a permutation)
$(\forall \pi :\ \pi \in Aut\ A :\ \pi(A), \pi(q^0) \models \bigwedge_{j:j\neq\pi(0)} g(\pi(0), j))$
iff ($\pi(A) = A$ as $\pi \in Aut\ A$, $\pi(q^0) = q^0$, as q^0 is symmetric)
$(\forall \pi :\ \pi \in Aut\ A :\ A, q^0 \models \bigwedge_{j:j\neq\pi(0)} g(\pi(0), j))$
$\Rightarrow$ (as $\mathcal{C}_n \leq Aut\ A$)
$(\forall \pi :\ \pi \in \mathcal{C}_n :\ A, q^0 \models \bigwedge_{j:j\neq\pi(0)} g(\pi(0), j))$
$\Rightarrow$ (The cyclic shifts in $\mathcal{C}_n$ drive 0 to i for every $i \in [n]$)
$(\forall i \in [n] : A, q^0 \models \bigwedge_{j:j\neq i} g(i,j))$
iff (by definition)
$A, q^0 \models \bigwedge_{i,j:i\neq j} g(i,j)$
$\square$

From this lemma, we need to consider only the pairs of processes $(0, j)$, such that $j \neq 0$. Applying 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. $\square$

Corollary 1 *For any $n \geq 4$, and a formula $g(i,j)$ of* $\mathsf{CTL}^*\backslash\mathsf{X}$,

1. $M(n), q_n^0 \models g(0,1)$ *iff* $M(4), q_4^0 \models g(0,1)$,
2. $M(n), q_n^0 \models g(0,n-1)$ *iff* $M(4), q_4^0 \models g(0,3)$, *and*
3. $M(n), q_n^0 \models g(0,j)$ *iff* $M(4), q_4^0 \models g(0,2)$, *for* $j \notin \{0, 1, n-1\}$.

Proof. The corollary follows from Theorems 1 and 5. $\square$

The main theorem results from applying this corollary:

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

Proof.

For any $n \geq 4$,
$M(n), q_n^0 \models f$
iff (the initial state is symmetric, lemma 3)
$M(n), q_n^0 \models \bigwedge_{j:j\neq 0} g(0,j)$
iff (splitting up the formula)
$M(n), q_n^0 \models g(0,1) \wedge M(n), q_n^0 \models g(0,n-1) \wedge$
$M(n), q_n^0 \models \bigwedge_{j:j\notin\{0,1,n-1\}} g(0,j)$
iff (by corollary 2)
$M(4), q_4^0 \models g(0,1) \wedge M(4), q_4^0 \models g(0,3) \wedge$
$M(4), q_4^0 \models g(0,2)$
iff (recombining the formula)
$M(4), q_4^0 \models \bigwedge_{j:j\neq 0} g(0,j)$
iff (by lemma 3)
$M(4), q_4^0 \models f$
$\square$

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

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

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

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, $A, q^0 \models \bigwedge_{i,j:i\neq j} g(i, i+1, j)$ iff $A, q^0 \models \bigwedge_{j:j\neq 0} g(0,1,j)$.*

Proof.

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

$(LHS \Leftarrow RHS)$

$A, q^0 \models \bigwedge_{j:j\neq 0} g(0,1,j)$
$\Rightarrow$ (π is a permutation)
$(\forall \pi:\ \pi \in Aut\ A:\ \pi(A), \pi(q^0) \models \bigwedge_{j:j\neq\pi(0)} g(\pi(0), \pi(1), j))$
iff ($\pi(A) = A$ as $\pi \in Aut\ A$, $\pi(q^0) = q^0$, as q^0 is symmetric)
$(\forall \pi:\ \pi \in Aut\ A:\ A, q^0 \models \bigwedge_{j:j\neq\pi(0)} g(\pi(0), \pi(1), j))$
$\Rightarrow$ (as $\mathcal{C}_n \leq Aut\ A$)
$(\forall \pi:\ \pi \in \mathcal{C}_n:\ A, q^0 \models \bigwedge_{j:j\neq\pi(0)} g(\pi(0), \pi(1), j))$
$\Rightarrow$ (The cyclic shifts in $\mathcal{C}_n$ drive 0 to i for every $i \in [n]$)
$(\forall i \in [n] : A, q^0 \models \bigwedge_{j:j\neq i} g(i, i+1, j))$
iff (by definition)
$A, q^0 \models \bigwedge_{i,j:i\neq j} g(i, i+1, j)$
$\square$

From this lemma, we need to consider only the triples of process indices of the form $(0, 1, j)$, such that $j \neq 0$. Application of the Reduction Theorem leads to the following theorem.

Theorem 7 *For any $n \geq 5$,*

1. $M(n)|_{(0,1)} \sim M(5)|_{(0,1)}$,
2. $M(n)|_{(0,1,2)} \sim M(5)|_{(0,1,2)}$,
3. $M(n)|_{(0,1,n-1)} \sim M(5)|_{(0,1,4)}$, *and*
4. $M(n)|_{(0,1,j)} \sim M(5)|_{(0,1,3)}$, *for $j \notin \{0,1,2,n-1\}$.*

Proof. Define $h : [n] \to [5]$ by $h(0) = 0$, $h(1) = 1$, $h(2) = 2$, $h(n-1) = 3$, and $h(j) = 3$, for $j \notin \{0,1,2,n-1\}$. 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. □

Corollary 2 *For any $n \geq 5$ and any formula $g(i, i+1, j)$ of $\mathsf{CTL}^*\backslash\mathsf{X}$,*

1. $M(n), q_n^0 \models g(0,1,1)$ *iff* $M(5), q_5^0 \models g(0,1,1)$,
2. $M(n), q_n^0 \models g(0,1,2)$ *iff* $M(5), q_5^0 \models g(0,1,2)$,
3. $M(n), q_n^0 \models g(0,1,n-1)$ *iff* $M(5), q_5^0 \models g(0,1,4)$, *and*
4. $M(n), q_n^0 \models g(0,1,j)$ *iff* $M(5), q_5^0 \models g(0,1,3)$, *for $j \notin \{0,1,2,n-1\}$.*

Proof. The corollary follows from Theorems 1 and 7. □

Theorem 8 *Let f be a formula of the form $\bigwedge_{i,j:i\neq j} g(i,i+1,j)$, where $g(i,i+1,j)$ is a $\mathsf{CTL}^*\backslash\mathsf{X}$ formula. Then, $(\forall n : n \geq 5, M(n), q_n^0 \models f \;\equiv\; M(5), q_5^0 \models f)$.* □

Proof.

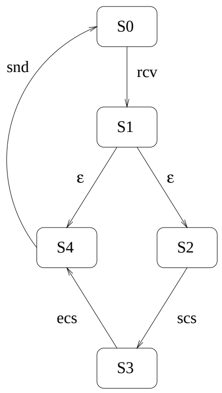
For any $n \geq 5$,
$M(n), q_n^0 \models f$
iff (the initial state is symmetric, lemma 4)
$M(n), q_n^0 \models \bigwedge_{j:j\neq 0} g(0,1,j)$
iff (splitting up the formula)
$M(n), q_n^0 \models g(0,1) \wedge M(n), q_n^0 \models g(0,1,2) \wedge$
$M(n), q_n^0 \models g(0,1,n-1) \wedge M(n), q_n^0 \models \bigwedge_{j:j\notin\{0,1,2,n-1\}} g(0,j)$
iff (by corollary 2)
$M(5), q_5^0 \models g(0,1) \wedge M(5), q_5^0 \models g(0,1,2) \wedge$
$M(5), q_5^0 \models g(0,1,4) \wedge M(5), q_5^0 \models g(0,1,3)$
iff (recombining the formula)
$M(5), q_5^0 \models \bigwedge_{j:j\neq 0} g(0,1,j)$
iff (by lemma 4)
$M(5), q_5^0 \models f$
□

5 Applications

To illustrate the use of the results, we look at two protocols, one for distributed mutual exclusion protocol [WL 89] and (a slight variation on) Milner's scheduler [Mil 90].

5.1 Distributed Mutual Exclusion

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



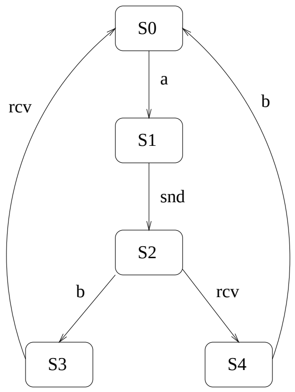
Initially, every process is in state $S0$. Here ϵ is used to indicate a local action. It is easily checked that the process description satisfies the conditions for the token passing model (cf. Section 3). So properties such as

- If a process requests the token, it will receive it. $\bigwedge_i \mathsf{AG}(S0_i \Rightarrow \mathsf{AF}(S1_i))$
- Every trying process eventually enters its critical section.
  $\bigwedge_i \mathsf{AG}(S0_i \Rightarrow \mathsf{AF}(S2_i \vee S4_i))$

can be checked using a 2-process system by theorem 3. Mutual exclusion, i.e $\bigwedge_{i,j:i \neq j} \mathsf{AG}\neg(S3_i \wedge S3_j)$ 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 $S3$. 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 8.

  $\bigwedge_{i,j:i \neq j} \mathsf{AG}\ (a_i \Rightarrow \mathsf{AXA}((\neg a_i \wedge \neg a_j)\ \mathsf{U}\ a_{i+1}))$
  $\bigwedge_{i,j:i \neq j} \mathsf{AG}\ (\neg(a_i \wedge a_j))$

- 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.

  $\bigwedge_i \ \mathsf{AG}((a_i \Rightarrow \neg a_i \ \mathsf{U} \ b_i) \ \wedge (b_i \Rightarrow \neg b_i \ \mathsf{U} \ a_i))$

These formulas are defined with atomic propositions referring to actions instead of states, but it is easily seen that the results apply to such formulas as well, as the block bisimulation of Theorem 2 matches computation paths with respect to non-τ actions.

6 Undecidability on Rings

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. W.l.o.g., we 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 this value domain to a binary one by encoding each of these values in unary. A processes passes one of these values to its neighbor by sending its unary code to the neighbor as a sequence of tokens with value 1. The neighbor accumulates the unary count on receiving such tokens, but retransmits them with value 0, which returns to the code sender, without effecting any change on the local states of the processes in between. This unary transmission is terminated by sending a token with value 0 to the neighbor.

Consider the property $\bigwedge_i (\textit{initially\_token}_i \Rightarrow \mathsf{AG}\neg \textit{halt}_i)$, 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.

The closest results are those 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. This has recently been extended to show global safety properties in [CGJ 95].

The advantage of the approach presented here is that, to check whether a ring comprised of isomorphic processes satisfies a specification for all instances, it is both necessary and sufficient to check only the small rings of size less than or equal to the cutoff. This result is independent of the actual program executing on 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.

An interesting problem to consider is whether there are conditions under which a similar result holds for systems with multiple-valued tokens. The simplicity of the 2-counter machine simulation suggests that imposing syntactic restrictions will not lead to decidability, unless coupled with some semantic constraints.

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.

[Br 89] Browne, M. Automatic Verification of Finite State Machines using Temporal Logic. PhD thesis, Carnegie-Mellon Univ., CMU-CS-89-117.

[C 93] Cleaveland, R. Analyzing Concurrent Systems using the Concurrency Workbench. in *Functional Programming, Concurrency, Simulation, and Automated Reasoning*, Springer-Verlag LNCS 693.

[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.

[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.

[dNV 90] De Nicola, R., Vaandrager, F. Three logics for Branching Bisimulation, 5th Annual IEEE Symp. on Logic in Computer Science, pp. 118-129, 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 82] Emerson, E.A., Halpern, J. Y. Decision procedures and Expressiveness in the Temporal Logic of Branching Time, *Proc. 14th ACM STOC*, San Francisco, 1982, pp. 169-180.

[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.

[ES 95] Emerson, E.A., Sistla, A.P. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic approach. CAV 1995.

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

[GW 89] van Glabbeek, R. J., Weijland, W. P. Branching time and abstraction in bisimulation semantics. in *Information Processing 89*, Elsevier Science Publishers, North-Holland, 1989.

[ID 93] Ip, C., Dill, D. Better verification through symmetry. Proc. 11th Intl. Symp. on Computer Hardware Description Languages and their Applications.

[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.

[La 80] Lamport, L. "Sometimes" is Sometimes "Not Never". in *POPL*, 1980.

[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.

[LP85] Litchtenstein, O., and Pnueli, A., Checking That Finite State Concurrent Programs Satisfy Their Linear Specifications, POPL 85, pp. 97-107.

[Lo 93] Long, D. Model Checking, Abstraction, and Compositional Verification. Ph.D. Thesis, Carnegie-Mellon University, 1993.

[Lu 84] Lubachevsky, B. An Approach to Automating the Verification of Compact Parallel Coordination Programs I. *Acta Informatica* 21, 1984.

[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.

[dNV 90] de Nicola, R., Vaandrager, F. Three logics for branching bisimulation. in *LICS*, 1990. Full version in *Journal of the ACM*, 42(2):458-487, 1995.

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

[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.

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

[Va 87] Vardi, M. Verification of Concurrent Programs - The Automata Theoretic Framework. in *LICS*, 1987. Full version in *Annals of Pure and Applied Logic*, 51:79-98, 1991.

[Va9?] Vardi, M. An Automata-theoretic Approach to Linear Temporal Logic, Proceedings of Banff Higher Order Workshop on Logics for Concurrency, F. Moller, ed., Springer-Verlag LNCS, to appear.

[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 *Let $I \subseteq [n]$, and $J \subseteq [k]$ be sets of indices such that there is a bijection $h : I \to J$ 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 $M(n)|_I$ is block equivalent to $M(k)|_J$.

Proof. Let h^A be the derived renaming function for indexed actions from $M(n)$ defined by $h^A(a_i) = a_{h(i)}$. Let h^P be the derived renaming function for indexed propositions from $M(n)$ defined by $h^P(P_i) = P_{h(i)}$. We define a relation that is a block bisimulation on $M(n)$ and $M(k)$ w.r.t. h^P.

For a state s of $M(i)$, let $tok_i(s)$ be the index of the process that possesses the token in state s. Let R_{nk} be the relation between states of $M(n)$ and $M(k)$, defined as follows:

$s\ R_{nk}\ t$ iff

1. The local states of D_n in s and D_k in t are identical,
2. $h(\lambda(s)|_I) = \lambda(t)|_J$,
3. $(\forall i : i \in I : tok_n.s = i \ \equiv\ tok_k.t = h.i)$, and
4. $(\forall i, j : i, j \in I : tok_n(s) \in (i : j)_n \ \equiv\ tok_k(t) \in (h(i) : h(j))_k)$.

The relation R_{nn} is similar but for (2), which is modified to $\lambda(s)|_I = \lambda(t)_I$. Define the relation $\approx$ to be $R_{nn} \cup R_{kk}$. Let $B = R_{nk} \cup R_{kn}$. B is symmetric. It is straightforward to check that $\approx$ is an equivalence relation, and that B is monotonic w.r.t. $\approx$. We show below that B is a block bisimulation w.r.t. $(\approx, h^P)$.

Suppose sBt and s is a state in $M(n)$, and t is a state in $M(k)$. By definition of B, $h^P(\lambda(s)|_I) = \lambda(t)|_J$,

1. s is the initial state of $M(n)$.

The only enabled action is the token transfer to some process i. As sBt, 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 $MAXF_{\approx}(s)$.

2. s is not the initial state of $M(n)$.

(a) Let $\sigma; v$ be a sequence starting at s, that is in $MAXF_{\approx}(s)$. Hence $v \not\approx s$, and by the definition of $\approx$, this must be because of either a local move by some process in I, or a token sent from some process in I, or because of a token received by some process in I.

- The action is an internal move, or a send of the token, by process i. As process $h(i)$ has the same local state, the same move can be performed at that process, and the resulting states are related by B. Note that the token is at process i iff it is at process $h(i)$ in t. This creates the matching sequence $\delta; w$.
- The action is a receipt of the token by process i from process $i-1$, for $i \in I$ (arithmetic is modulo n). As sBt, the token in t is not at process $h(i)$. Hence, if process $h(i) - 1$ is ready to send the token, the receive action may be executed at process $h(i)$.

  Otherwise, the token is in the interval $(h(j) : h(i))$, for some j. As the process with the token must eventually reach a blocking send action, and processes without the token reach a blocking receive action, there is a sequence of events, δ, starting at state t, such that after δ, the token is at process $h(i) - 1$, which is blocked on a send action. At this point, the token may be transferred to process $h(i)$ and the system moves to state w. Note that every state along δ is related to t by $\approx$, and $w \not\approx t$; hence, $\delta; w$ is a member of $MAXF_{\approx}(t)$. The final state w is related to v.

(b) The other case, that of infinite sequences from s that are in $MAXI_{\approx}(s)$, 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 case. It follows that B is a block bisimulation w.r.t. $(\approx, h^P)$ between $M(n)|_I$ and $M(k)|_J$. Notice that this proof shows also that the non-τ actions are matched by B up to h. $\square$

A.2 Block Bisimulation

Let A be an LTS, $A = (Q, \Sigma, \delta, \lambda, L, q^0)$. [BCG 88] define stuttering bisimulation only over finite state systems. Their definition can be generalized to the following :

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 σ starting at s, there is a fullpath δ starting at t such that σ matches δ by B.*

where fullpaths σ and δ match by B iff they can be partitioned into an equal number of segments such that, for each segment number i, every state in the ith segment of σ is related by B to every state in the ith segment of δ. The formal definition of "match" is given below :

Matching paths

Let INC be the set of strictly increasing sequences of natural numbers starting at 0. Precisely, $INC = \{\pi \mid \pi : \mathbf{N} \to \mathbf{N} \wedge \pi(0) = 0 \wedge (\forall i\ i \in \mathbf{N} \Rightarrow \pi(i) < \pi(i+1))\}$. Let σ be a path, and π a member of INC. For $i \in \mathbf{N}$, let $intv(i, \sigma, \pi) = [\pi(i), min(\pi(i+1), length(\sigma)))$. The ith segment of σ w.r.t. π, $seg(i, \sigma, \pi)$, is defined by the sequence of states of σ with indices in $intv(i, \sigma, \pi)$.

Let σ and δ, under partitions π and ξ respectively, correspond w.r.t. stuttering bisimulation B iff they are subdivided into the same number of segments, and any pair of states in segments with the same index are related by B. Precisely,
$corr_B((\sigma, \pi), (\delta, \xi)) \equiv (\forall i : i \in \mathbf{N} : intv(i, \sigma, \pi) \neq \emptyset \equiv intv(i, \delta, \xi) \neq \emptyset \wedge (\forall m, n : m \in intv(i, \sigma, \pi) \wedge n \in intv(i, \delta, \xi) : (\sigma_m, \delta_n) \in B))$.

Paths σ and δ match iff there exist partitions that make them correspond. Precisely, $match_B(\sigma, \delta) \equiv (\exists \pi, \xi : \pi, \xi \in INC : corr_B((\sigma, \pi), (\delta, \xi)))$. □

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 be a block bisimulation w.r.t. $(\approx, id)$. Then B is also a stuttering bisimulation.*

Proof. B is symmetric by (1) of Defn. 3. Let (s, t) be an arbitrary pair in B. Then, $\lambda(s) = \lambda(t)$ by 3(a). Consider any fullpath σ starting at s. If $\sigma \in MAXI_{\approx}(s)$, then by 3(c) of Defn. 3, there is a fullpath δ starting at t such that $\delta \in MAXI_{\approx}(t)$, which is infinite if and only if σ is infinte. So σ and δ may be partitioned into the same number of segments, which by (2), consist of states related by B. If $\sigma \notin MAXI_{\approx}(s)$, there is a maximal prefix σ' of σ with a following state v such that $\sigma; v \in MAXF_{\approx}(s)$. By 3(b), there is a finite path $\delta'; v$ in $MAXF_{\approx}(t)$ such that $(u, v) \in B$. σ' and δ' form a pair of matching segments, and (u, v) the start of a new pair of corresponding segments. Continuing in this manner, one can inductively define a fullpath δ starting at t, and partitions of σ and δ such that σ and δ match w.r.t. B. Hence, B is a stuttering bisimulation. □

Theorem 4 *Let B be the greatest stuttering bisimulation. Then B is a block bisimulation w.r.t. (B, id).*

Proof. The greatest stuttering bisimulation is an equivalence relation; hence conditions (1) and (2) of the block bisimulation definition are satisfied. Let (s, t) be an arbitrary pair in B. As $\lambda(s) = \lambda(t)$, by (1) of Defn. 5, it follows that $id(\lambda(s)) = \lambda(t)$.

Consider a path σ in $MAXI_B(s)$. As σ is a fullpath, by (2) of Defn. 5, there is a fullpath δ starting at t such that σ matches δ w.r.t. B. This implies that δ is in $MAXI_B(t)$ as B is an equivalence, and δ satisfies condition 3(c) of Defn. 3.

Consider a finite path $\sigma;v$ in $MAXF_B(s)$. Let γ be a fullpath starting at v. As $\sigma;\gamma$ is a fullpath starting at s, by 2(b) of Defn. 5, there is a matching fullpath δ starting at t. As $\neg(vBs)$, $\neg(vBt)$ holds. Let i be the first position along δ such that $vB\delta_i$ holds. Hence, $i > 0$, and $\delta[0..i)$ consists of states related to t by B. So $\delta[0..i]$ is a member of $MAXF_B(t)$, and satisfies condition 3(b) of Defn. 3. Hence, B is a block bisimulation w.r.t. (B, id). $\square$

A.3 Proof of Theorem 1

Theorem 1 *Let B be a block bisimulation on LTS A w.r.t. $(\approx, h)$. If $(s,t) \in B$, then for any formula f of* $\mathsf{CTL}^*\backslash\mathsf{X}$, *$A, s \models f$ iff $A, t \models h(f)$.* $\square$

Proof.

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

Basis : $f \in AP$.

$M, s \models f$
iff (definition)
$f \in \lambda(s)$
iff (h is a bijection on AP)
$h(f) \in h(\lambda(s))$
iff (condition 3(a) of Defn. 3)
$h(f) \in \lambda(t)$
iff (definition)
$M, t \models h(f)$

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.

$M, s \models \mathsf{E}g$
iff (definition)
$(\exists\sigma : fp(\sigma) \wedge \sigma_0 = s : M, \sigma \models g)$
iff (by following proof)
$(\exists\delta : fp(\delta) \wedge \delta_0 = t : M, \delta \models h(g))$
iff (definition)
$M, t \models \mathsf{E}h(g)$
iff (definition)
$M, t \models h(f)$

To justify the second step of the proof above, suppose σ is a fullpath starting at s. As $(s,t) \in B$, by Defn. 3 and a simple inductive argument involving 3(b) and 3(c), there is a fullpath δ starting at t such that σ and δ may be partitioned into the same number of nonempty segments, where the initial states of corresponding segments are related by B, and states within each segment are in the same $\approx$-class. Since initial states of corresponding segements are related by B, from the monotonicity of B w.r.t. $\approx$ (property (2)), every pair of states from corresponding segments is in B. Hence, by the inductive claim, states related by B agree on the truth value of sub-formulas of g.

Let σ and δ be such that they can be partitioned as described above. The nested inductive claim is that for any path formula g such that states related by B agree on the truth values of sub-formulas of g modified by h, $M, \sigma \models g$ iff $M, \delta \models h(g)$.

Basis : g is a state formula. As $\sigma_0 B \delta_0$, the claim follows from the nested induction hypothesis.

Inductive case : The proof when g has the form $g_0 \wedge g_1$ or $\neg g_0$ follows directly from the induction hypothesis. Consider the case where g has the form $g_0 \mathsf{U} g_1$.

$M, \sigma \models g_0 \mathsf{U} g_1$
iff (definition)
$(\exists i : M, \sigma^i \models g_1 \wedge (\forall j : j < i : M, \sigma^j \models g_0))$
iff (σ and δ have matching partitions; inductive hypothesis)
$(\exists k : M, \delta^k \models h(g_1) \wedge (\forall l : l < k : M, \sigma^l \models h(g_0)))$
iff (definition)
$M, \delta \models h(g_0) \mathsf{U} h(g_1)$
iff (definition)
$M, \delta \models h(g_0 \mathsf{U} g_1)$

The second step in the proof follows from the observation that since the paths match, for any position i on one path, in the first position k on the corresponding segment of the other path is such that the suffix paths starting at i and k match, and for each suffix path starting at position $l < k$, there is a matching path from some position $j < i$. Hence, the induction hypothesis can be applied to these suffixes. $\square$