Symmetry and Model Checking

E. Allen Emerson* A. Prasad Sistla†

November 11, 1994

"Whenever you have to do with a structure-endowed entity Σ try to determine its group of automorphisms"
– Hermann Weyl
in *Symmetry*

Abstract

We show how to exploit symmetry in model checking for concurrent systems containing many identical or isomorphic components. We focus in particular on those composed of many isomorphic processes. In many cases we are able to obtain significant, even exponential, savings in the complexity of model checking.

1 Introduction

In this paper, we show how to exploit symmetry in model checking. We focus on systems composed of many identical (isomorphic) processes. The global state transition graph $\mathcal{M}$ of such a system exhibits a great deal of symmetry, characterized by the group of graph automorphisms of $\mathcal{M}$. The basic idea underlying our method is to reduce model checking over the original structure $\mathcal{M}$, to model checking over a smaller quotient structure $\overline{\mathcal{M}}$, where symmetric states are identified. In the following paragraphs, we give a more detailed but still informal account of a "group-theoretic" approach to exploiting symmetry.

More precisely, the symmetry of $\mathcal{M}$ is reflected in the group, *Aut* $\mathcal{M}$, of permutations of process indices defining graph automorphisms of $\mathcal{M}$. Similarly, any specification formula f intended to capture correctness of $\mathcal{M}$ in a particular Temporal Logic (say, CTL*) exhibits a certain degree of "internal" symmetry reflected in the group, *Auto* f, of permutations of process indices that leave f and significant subformulas of f invariant.

We show that for any group G contained in *Aut* $\mathcal{M}$, we can define $\overline{\mathcal{M}} = \mathcal{M}/G$ to be the quotient structure obtained by identifying any two states s, t of $\mathcal{M}$ that are in the same orbit (or equivalence class) of the state space of $\mathcal{M}$ induced by G in the usual way: there exists a permutation π in G such that $\pi(s) = t$. In other words, s and t are the same except for a permutation of their indices. (For example: $s = (N_1, T_2, C_3), t = (N_2, T_3, C_1)$).

[1]Department of Computer Sciences, The University of Texas at Austin, USA. The author's work was supported in part by ONR contract N00014-89-J-1472, Semiconductor Research Corporation Contract 94-DP-388, and Texas Advanced Technology Program Grant 003658-250.

[1]Department of Electrical Engineering and Computer Science, The University of Illinois at Chicago, USA. The author's work was supported in part by NSF grant CCR-9212183.

We next show that such a quotient structure $\overline{\mathcal{M}}$ corresponds in a coarse sense to the original structure $\mathcal{M}$, so that if there is a path in $\overline{\mathcal{M}}$ there is an analogous path in $\mathcal{M}$, and conversely. However, the correspondence may not be sufficiently precise to (directly) model check a specification f. If we further stipulate that G be contained in $Aut\ \mathcal{M}\ \cap\ Auto\ f$ then we get a precise correspondence enabling us to establish

$$\mathcal{M}, s \models f \quad \text{iff} \quad \overline{\mathcal{M}}, \overline{s} \models f$$

where f is a formula of CTL* or Mu-Calculus, and $\overline{s}$ indicates the equivalence class of s.

We emphasize here that any subgroup G of $Aut\ \mathcal{M} \cap Auto\ f$ is sufficient. If we take $G = Aut\ \mathcal{M}\ \cap\ Auto\ f$ then we get maximal compression. However, determination of this G seems to be a potentially difficult problem. This is due to the fact that the problem of computing $Aut\ \mathcal{M}$ is polynomial time equivalent to graph isomorphism (cf. [Ho82]). Fortunately, since $\mathcal{M}$ is derived from a concurrent system $\mathcal{P} = //_i K_i$ consisting of many isomorphic processes K_i, we are able to show that $Aut\ CR\ \subseteq\ Aut\ \mathcal{M}$, where CR is the process communication graph for $\mathcal{P}$. Since CR often follows a simple, standard pattern, $Aut\ CR$ is often known in advance, and we can use $G = Aut\ CR\ \cap\ Auto\ f$. Moreover, for massively parallel architectures $Aut\ CR$ is likely to be a large group reflecting a high degree of symmetry. Determination of $Auto\ f$ automatically is also a difficult problem. However, $Auto\ f$ can often be determined manually by examination of the formula.

For many of the automorphism groups G determined in practice we can efficiently and incrementally compute $\mathcal{M}/G$, there by circumventing the construction of $\mathcal{M}$. Of course, we then accrue the advantage of model checking over the smaller structure $\overline{\mathcal{M}} = \mathcal{M}/G$.

One common and advantageous case occurs when $G = Sym\,[1:n]$, the set of all permutations on indices $[1:n]$. For a system with n processes each with l local states, the original structure can have on the order of l^n states, while $\mathcal{M}/G$ has on the order of n^l states. When l is fixed and relatively small, while n is large, then $n^l \ll l^n$. We can thus realize exponential savings.

A complication can occur when f is a complex formula with little symmetry. Then $Auto\ f$ and hence G may be small, resulting in little compression. We argue that it is frequently beneficial to decompose f into smaller constituent subformulae and check those individually. We also show how the symmetry of individual states can be exploited for further gains in efficiency.

Finally, we give an alternative, automata-theoretic approach that provides a uniform method permitting the use of a *single* quotient $\overline{\mathcal{M}} = \mathcal{M}/Aut\ \mathcal{M}$ for model checking for many specifications f, without computing and intersecting with $Auto\ f$. The idea is to annotate the quotient with "guides", indicating how coordinates are permuted from one state to the next in the quotient. An automaton for f designed to run over paths through $\mathcal{M}$, can be modified into another automaton run over $\overline{\mathcal{M}}$ using the guides to keep track of shifting coordinates.

The remainder of the paper is organized as follows: in section 2 we give preliminary definitions and terminology. In section 3 we describe our group-theoretic approach showing that, for both CTL* and the Mu-calculus, model checking over the original structure can be reduced to model checking over the quotient structure $\mathcal{M}/G$ for any G which is a subgroup of $Aut\ \mathcal{M}\ \cap\ Auto\ f$. In section 4 we discuss how the method can be applied in practice. This includes showing a helpful way to approximate $Aut\ \mathcal{M}$ from the network topology CR, by establishing that $Aut\ CR\ \subseteq\ Aut\ \mathcal{M}$. We also discuss optimizations based on formula decomposition and state symmetry. An alternative automata-theoretic approach using annotated quotient structures is described in section 5. An examples is given in section 6. In the section 7 we discuss related work, and we give concluding remarks in section 8.

2 Preliminaries

2.1 Model of Computation

We deal with *structures* of the form $\mathcal{M} = (\mathcal{S}, \mathcal{R})$ where

- $\mathcal{S} = L^I \times D^V$ is the finite set of *states*, with L a finite set of individual process *locations*, I the set of process indices, and V is a finite set of shared *variables* over a finite *data domain* D.[1]

- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ which represents the moves of the system.

Notation: For convenience, each state $s = (s', s'') \in \mathcal{S}$ can be written in the form $(\ell_1, \ldots, \ell_n, v = d, \ldots, v' = d')$ indicating that processes $1, \ldots, n$ are in locations $\ell_1, \ldots, \ell_n$, respectively and the shared variables $v, \ldots, v'$ are assigned data values $d, \ldots, d'$, respectively.

As usual, a *path* through $\mathcal{M}$ is a finite or infinite sequence of states such that every consecutive pair of states is in $\mathcal{R}$. By a convenient abuse of notation, we denote a path by $s_0, s_1, s_2, \ldots$ or by $s_0 \longrightarrow s_1 \longrightarrow s_2, \ldots$ not bothering to explicitly indicate the last state for finite paths. A *fullpath* is a maximal path, i.e., either an infinite path or a finite one whose last state lacks an $\mathcal{R}$-successor.

In practice, for ordinary model checking, $\mathcal{M}$, is the global state transition graph of a finite state concurrent program $\mathcal{P}$ of the form $//_i K_i$ consisting of processes $K_1, \ldots, K_n$ running in parallel. Each K_i may be viewed as a finite state transition graph with node set L. An arc from node ℓ to node ℓ' may be labelled by a guarded command $B \longrightarrow A$. The guard B is a predicate that can inspect shared variables and local states of "accessible" processes. The action A is a set of simultaneous assignments to shared variables $v := d \parallel \cdots \parallel v' := d'$.[2] When process K_i is in local state ℓ and the guard B evaluates to *true* in the current global state, the global system can nondeterministically choose to advance by firing this transition of K_i which changes the local state of K_i to be ℓ' and the shared variables in V according to A. Thus the arc from ℓ to ℓ' in K_i represents a *local transition* of K_i that we denote by $\ell : B \longrightarrow A : \ell'$.

The structure $\mathcal{M}$ corresponding to $\mathcal{P}$ is thus defined using the obvious formal operational semantics. First, the set of (all possible) states $\mathcal{S}$ is determined from $\mathcal{P}$ because it provides us with the set of local (i.e., individual process) locations L, process indices I, variables V, and data domain D. For states $s, t \in \mathcal{S}$, we define $s \longrightarrow t \in \mathcal{R}$ iff
$\exists i \in I$ process K_i can cause s to move to t, denoted $s \longrightarrow_i t$ iff
$\exists i \in I$ $\exists$ local transition $\tau_i = \ell_i : B_i \longrightarrow A_i : m_i$ of K_i which *drives* $s = (s', s'')$ to $t = (t', t'')$; this means the i-th component of s' equals ℓ_i, the i-th component of t' equals m_i, all other components of s' equal the corresponding component of t', predicate $B_i(s) = true$, and $t'' = A_i(s'')$.

We are often interested in just the set of states reachable by executing $\mathcal{P}$ starting in a particular start state s_0. It is often most natural to consider execution of a program appropriately initialized. Moreover, the set of states reachable from s_0 can be much smaller than the set of all possible states. It is thus important to note that we can incrementally generate the (initialized) structure $\mathcal{M} = (\mathcal{S}, \mathcal{R}, s_0)$ corresponding to $\mathcal{P}$ starting in state s_0. We use the notation $K_i(s)$ to denote the set of

[1] We remark that D and V are optional, in which case we define $\mathcal{S} = L^I$. When present, D and V can have their own additional internal organization. In particular, they can depend on I.

[2] We stipulate that each guarded command be *index independent*, which means that the value of the guard and the effect of the action do not depend on the specific values chosen for the index set I. In particular, permuting the names of the indices should not alter their values. This excludes, for example, such guards as $1 < 3$, whose truth value would change under transposition of 1 and 3.

states reachable from state s by a single step of process K_i. We begin with s_0, propagate it by adding in the members of the various $K_i(s_0)$'s, and then propagate the K_i's of those members, and so on until closing off. See section 4.2 for a helpful and important generalization of this idea.

2.2 Logics of Programs

We assume a familiarity with basic aspects of temporal and modal logics of programs (cf. [Em91], [MP92], [St93]). In this paper we use the logic CTL* and the Mu-calculus.

2.2.1 CTL*. The logic CTL* uses the temporal operators U (until), X (nexttime) and the existential (full-)path quantifier E. The set of CTL* (path) formulas is generated by the following rules:

- every atomic proposition, such as P, is a CTL* formula
- if g, h are CTL* formulas then $g \mathsf{U} h$, $\mathsf{E}g$, $\mathsf{X}g$, $g \wedge h$ and $\neg h$ are also CTL* formulas.

We write $\mathcal{M}, x \models f$ to denote that in structure $\mathcal{M}$ of fullpath $x = (x_0, x_1, ...)$ formula f is true; the definition of $\models$ is specified inductively:

- $\mathcal{M}, x \models g \mathsf{U} h$ iff for some $i \geq 0$, $\mathcal{M}, x^{(i)} \models h$ and for all j, such that $0 \leq j < i$, $\mathcal{M}, x^{(j)} \models g$, where $x^{(i)}$ denotes the suffix of x starting from x_i.
- $\mathcal{M}, x \models \mathsf{E}g$ iff there exists a maximal path x' starting from x_0, which may be different from x, such that $\mathcal{M}, x' \models g$.
- $\mathcal{M}, x \models \mathsf{X}g$ iff $\mathcal{M}, x^{(1)} \models g$.
- $\mathcal{M}, x \models g \wedge h$ iff $\mathcal{M}, x \models g$ and $\mathcal{M}, x \models h$.
- $\mathcal{M}, x \models \neg g$ iff it is not the case that $\mathcal{M}, x \models g$.
- $\mathcal{M}, x \models P$ iff P is true in the state x_0, for any atomic proposition P.

Convention: Indexed atomic propositions (cf. [CG89]) and atomic formulas are treated as follows. If ℓ is a local process state and some process i is in local state state ℓ in global state s, then s is of the form $(\ldots, \ell_i, \ldots)$ and we say that indexed proposition ℓ_i is *true* in global state s. If variable v has value d in global state s, then s is of the form $(\ldots, v = d, \ldots)$, and we say that the atomic formula $v = d$, which we treat as an atomic proposition, is *true* in global state s.

Any CTL* formula which is a boolean combination of atomic propositions and formulas of the form $\mathsf{E}g$ is called a *state formula*. Note that in a structure all fullpaths starting from the same state satisfy the *same* set of state formulas. We write $\mathcal{M}, s \models f$, and say that in structure $\mathcal{M}$ at state s formula f is true, provided that $\mathcal{M}, x \models f$ for all fullpaths x starting at s.

We find it convenient to use the other standard temporal operators F (sometime), G (always), and propositional connectives $\vee$ (or), $\Rightarrow$ (implies), A (universal path quantifier). All these operators and connectives can be defined in terms of the basic symbols in the usual way; e.g., $\mathsf{A}f$ abbreviates $\neg\mathsf{E}\neg f$ and $g \vee h$ abbreviates $\neg((\neg g) \vee (\neg h))$. The logic CTL (see [CES83]) is strict subset of CTL* which restricts how the temporal operators can be used with path quantifiers.

2.2.2 The Mu-calculus. We define the syntax and semantics of the (propositional) Mu-calculus (cf. [Ko83], [EC80]). We assume that we have a set $\mathcal{X}$ of variables whose members are denoted by $y, z, \ldots$. The formulas of the Mu-calculus are formed using (indexed) atomic propositions, variables, the propositional connectives $\neg$ and $\wedge$, the modal operator $\mathtt{<R>}$ and the least fixpoint operator μ, which is formally analogous to a quantifier. The set of formulas of the Mu-calculus is the smallest set satisfying the following properties:

- each atomic proposition P and each variable y in $\mathcal{X}$ is a formula
- if f and g are formulas then $f \wedge g$, $\neg f$, $\mathtt{<R>}f$ are also formulas
- if $f(y)$ is a formula, then $\mu y.f(y)$ is also a formula, provided all occurrences of the variable y in f are in the scope of an even number of negations

To define the semantics, we need the following terminology. A variable y is *free* in a formula f if there is at least one occurrence of y which is not in the scope any μy. The set of variables that are free in f is denoted by *free-var*(f). A formula without any free variables is called a *closed formula* or *sentence*. Let $\mathcal{M} = (\mathcal{S}, \mathcal{R})$ be a structure. A *valuation* ρ is a mapping that associates a subset of $\mathcal{S}$ with each variable in $\mathcal{X}$. With each structure $\mathcal{M}$ as given above and with each formula f, we define a function $\mathcal{L}_{(\mathcal{M},f)}$ from the set of valuations to subsets of $\mathcal{S}$, by induction on the structure of f as follows:

- $\mathcal{L}_{(\mathcal{M},P)}(\rho) = \{s \in \mathcal{S} : \mathcal{M}, s \models P\}$ where P is an atomic proposition
- $\mathcal{L}_{(\mathcal{M},y)}(\rho) = \rho(y)$
- $\mathcal{L}_{(\mathcal{M},f\wedge g)}(\rho) = \mathcal{L}_{(\mathcal{M},f)}(\rho) \cap \mathcal{L}_{(\mathcal{M},g)}(\rho)$
- $\mathcal{L}_{(\mathcal{M},\neg f)}(\rho) = \mathcal{S} - \mathcal{L}_{(\mathcal{M},f)}(\rho)$
- $\mathcal{L}_{(\mathcal{M},\mathtt{<R>}f)}(\rho) = \{s : \exists s' \in \mathcal{L}_{(\mathcal{M},f)}(\rho)$ such that $(s, s') \in \mathcal{R}\}$
- $\mathcal{L}_{(\mathcal{M},\mu y.f(y))}(\rho) = \bigcap\{S' \subseteq \mathcal{S} : S' = \mathcal{L}_{(\mathcal{M},f(y))}(\rho')$ where $\rho'(y) = S'$ and for all other $z \in \mathcal{X}$, $\rho'(z) = \rho(z)\}$.

Note that the value of $\mathcal{L}_{(\mathcal{M},\mu y.f(y))}(\rho)$ is given as a least fixed point. For finite Kripke structures, the least fixed point can be computed by starting with the empty set and iterating f at most $|\mathcal{S}|$ times until a fixed point is reached, by the well-known Tarski-Knaster theorem.

Other connectives can then be introduced as abbreviations: $\nu y.f(y)$ abbreviates $\neg\mu y.\neg f(\neg y)$ and represents the *greatest* fixpoint of $f(y)$, while $\mathtt{[R]}f$ abbreviates $\neg\mathtt{<R>}\neg f$. Other propositional connectives are defined as abbreviations in the usual way.

2.3 Applicable Group Theory

We summarize the essential notions from group theory needed here. We refer the reader to one of the many standard texts discussing this topic (cf. [He64]) for additional information. A group $\mathcal{G}$ is a set G together with a binary operation on G, called the group multiplication, that is associative, has an identity, and has an inverse for each group element. In practice, we write just G for $\mathcal{G}$, and multiplication may be indicated by concatenation. $H \leq G$ denotes H is a subgroup of G.

A *permutation* π on a finite set of objects I is a 1-1, onto mapping $\pi : I \longrightarrow I$. The set of all permutations on I, denoted $Sym\ I$, forms a group under functional composition: if permutations π',$\pi'' \in Sym\ I$ then $\pi = \pi'' \circ \pi' \in Sym\ I$. Here the order of functional composition in $\pi'' \circ \pi'$ is to first apply π' then apply π''. If $J \subseteq I$ then $Pstab\ J$ denotes $\{\pi : \forall\, j \in J\ \pi(j) = j\}$, the *pointwise stabilizer* of J. Id is the *identity* permutation or relation on I.

Given an indexed object b, i.e. one whose description depends on I, we can define a notion of *permutation* π being *applied to* b, denoted $\pi(b)$. In general, $\pi(b)$ is obtained from b by simultaneously replacing every occurrence of index $i \in I$ by $\pi(i)$.

For example, given state $s = (N_1, T_2, C_3, turn = 1)$, where $\{N, T, C\} \subseteq L$, $turn$ is a shared variable, and $\pi : 1 \mapsto 2, 2 \mapsto 1, 3 \mapsto 3$, we have $\pi(s) = (N_{\pi(1)}, T_{\pi(2)}, C_{\pi(3)}, turn = \pi(1)) = (N_2, T_1, C_3, turn = 2) = (T_1, N_2, C_3, turn = 2)$.

Roughly speaking, we can then define $Aut\ b$ to be the set (which is, in fact, a group) of permutations $\pi \in Sym\ I$ such that $\pi(b)$ is "equivalent" to b. The notion of equivalence used depends on the type of object b and the intended application.

2.4 Automorphisms of States

We define $Aut\ s = \{\pi \in Sym\ I : \pi(s) = s\}$ for any state $s \in \mathcal{S}$. Similarly, for any $\mathcal{T} \subseteq \mathcal{S}$ we define $Aut\ \mathcal{T} = \{\pi \in Sym\ I : \pi(\mathcal{T}) = \mathcal{T}\}$.

2.5 Automorphisms of a Structure

We will define a notion of automorphism h of structure $\mathcal{M}$ into itself. By analogy with the usual definition of graph automorphism for labeled, directed graphs we say the following:

An *automorphism* h of structure $\mathcal{M} = (\mathcal{S}, \mathcal{R})$ is a mapping $h : \mathcal{S} \longrightarrow \mathcal{S}$ that

1. is 1-1, onto on $\mathcal{S}$,
2. preserves edge structure: $s \longrightarrow t \in \mathcal{R}$ implies $h(s) \longrightarrow h(t) \in \mathcal{R}$, and
3. preserves "labeling" of states up to a permutation: $h(s) = \pi'(s)$ for some $\pi' \in Sym\ I$.

If $\mathcal{M} = (\mathcal{S}, \mathcal{R}, s_0)$ is initialized, we also require that

4. $h(s_0) = s_0$.

Observe, in particular, that a permutation π on I, viewed as a mapping $\mathcal{S} \longrightarrow \mathcal{S}$, vacuously satisfies the 1st and 3rd criteria. If it also fulfills the 2nd criterion then it is an automorphism of $\mathcal{M}$. We define $Aut\ \mathcal{M} = \{\pi \in Sym\ I : \pi$ defines an automorphism of $\mathcal{M}\}$. More simply, we have $Aut\ \mathcal{M} = \{\pi \in Sym\ I : \pi(\mathcal{M}) = \mathcal{M}\}$.

2.6 Automorphisms of formulas

For a CTL* formula f, we let $Aut\ f = \{\pi \in Sym\ I : \pi(f) \equiv f\}$, where $\equiv$ denotes logical equivalence under all propositional interpretations. For example, for $f = P_1 \wedge P_2$ and $\pi = Flip$, the permutation transposing 1 and 2, we have $\pi(f) = P_{\pi(1)} \wedge P_{\pi(2)} = P_2 \wedge P_1 \equiv P_1 \wedge P_2 = f$. Hence, $Aut\ f = \{Id, Flip\}$. In general, $Aut\ f$ is intended to capture the "top-level" symmetry of f.

We also use a subset (subgroup) of $Aut\ f$, denoted by $Auto\ f$ which is used to capture the "internal" symmetry of f and certain significant subformulas thereof. This internal symmetry will subsequently turn out to be vital to formulating inductive arguments on formula structure in proving the Compression Theorem below. $Auto\ f$ is defined as follows:

- For a propositional formula f, we define $Auto\ f = Aut\ f$.
- For a general CTL* formula f, we define $Auto\ f$ inductively according to the following cases.
  - $f = \mathsf{X}g$ or $f = \mathsf{E}g$: In this case, $Auto\ f = Auto\ g$.
  - $f = g\ \mathsf{U}\ h$: In this case, $Auto\ f = Auto\ g \cap Auto\ h$.
  - Other cases: If neither of the above conditions hold then f is a boolean combination of atomic propositions and subformulas of the form $\mathsf{X}g$, $g\ \mathsf{U}\ h$ and $\mathsf{E}g$. That is, $f = b(e_1, e_2, ..., e_k, f_1, f_2, ..., f_l)$ where b is a boolean formula over the atomic propositions $e_1, e_2, ..., e_k$ and subformulas $f_1, f_2, ..., f_l$ where each f_i is of the form $\mathsf{X}g$, or $g\ \mathsf{U}\ h$, or $\mathsf{E}g$. Now, we replace each f_i in b by a new unindexed proposition F_i, and define $Auto\ f = Auto\ b(e_1, e_2, ..., e_k, F_1, F_2, ..., F_l) \cap Auto\ f_1 \cap ...\ Auto\ f_l$. It is to be noted that $b(e_1, .., e_k, F_1, ..., F_l)$ is a propositional formula.

It is not difficult to see that $Auto\ f$ is well- defined for any CTL* formula. For example, letting $I = [1:2]$, consider $f = P_1 \wedge \mathsf{EX}(Q_1 \vee Q_2) \ \vee \ P_2 \wedge \mathsf{EX}(Q_1 \vee Q_2)$. ¿From the definitions we get $Auto\ f = Auto\ (P_1 \wedge B \ \vee \ P_2 \wedge B) \cap Auto\ (\mathsf{EX}(Q_1 \vee Q_2))$ where B is considered as unindexed proposition, while P_1 and P_2 are considered as indexed propositions. Now, $Auto\ (P_1 \wedge B \vee P_2 \wedge B) = Sym\ I$ and we also see that $Auto\ \mathsf{EX}(Q_1 \vee Q_2)$ is also $Sym\ I$, and hence $Auto\ f = Sym\ I$.

Remark: There is an alternate way of capturing internal symmetry of f. If $q_1, \ldots, q_m$ are the maximal propositional subformulae of f with respect to the subformula relation, then define $Auto'\ f = Aut\ q_1 \cap \ldots \cap Aut\ q_m$. $Auto'\ f$ consists of those permutations respecting the symmetry not only of f but also of its major constituent propositional subformulae q_i. It can be shown that the first definition, $Auto\ f$, is more general; i.e. for any f, the $Auto'\ f \subseteq Auto\ f$. In addition, for some formulas, the containment is strict. The formula f given in the previous paragraph is such an example. In the rest of the paper, we will only use $Auto\ f$.[3]

For a Mu-calculus formula f, $Auto\ f$ is defined inductively according to the following cases:

- $f = \mu y.g$ or $f = \texttt{<R>}g$: In this case, $Auto\ f = Auto\ g$.
- $f = y$: In this case, $Auto\ f = Sym\ I$.
- Other cases: In all other cases f can be written as a boolean combination of indexed atomic propositions and subformulas which are variables or of the form $\mu y.g$ or $\texttt{<R>}g$. That is, $f = b(e_1, ..., e_k, f_1, ..., f_l)$ where each e_i is an indexed atomic proposition and each f_j is a variable or a subformula of the form $\texttt{<R>}g$ or of the form $\mu y.g$. In this case, we define $Auto\ f = Auto\ b(e_1, ..., e_k, F_1, ..., F_l) \cap Auto\ f_1 \cap ... \cap Auto\ f_l$ where $F_1, ..., F_l$ are unindexed atomic propositions. Note that $b(e_1, ..., e_k, F_1, ..., F_l)$ is simply a propositional formula.

For example, if $f = \mu y.((P_1 \vee P_2) \vee \texttt{<R>}(Q_1 \wedge Q_2 \wedge \texttt{<R>}y))$ then $Auto\ f = Sym\ I$.

2.7 Quotient Construction

Finally, let G be any subgroup of $Sym\ I$. Then we can define an equivalence relation $\equiv_G$ on states in $\mathcal{S}$ where $s \equiv_G t$ iff $\exists\ \pi \in G$ such that $t = \pi(s)$. The equivalence class of s, denoted

[3]The definition of "$Auto\ f$" in [ES93] amounts to $Auto'\ f$ defined here. The Compression Theorems with the new, more general definition of $Auto\ f$ are also true when $Auto\ f$ is replaced by the "old" $Auto'\ f$ of [ES93]. The advantage of the new $Auto\ f$ is that, since it is in general a superset of the old, it may provide greater compression. Moreover, it should be noted that $Auto\ \Phi = Auto'\ \Phi = \{Id\}$ for Φ a fairness constraint (cf. [ES94]).

$[s]_G$, is also referred to as the *G-orbit* of s. In the sequel, our task will be to find a subgroup G of $Sym\ I$ that is a subgroup of $Aut\ \mathcal{M}$ thus respecting the symmetry of $\mathcal{M}$ and also is a subgroup of $Auto\ f$, thus respecting the symmetry of f. We then collapse *G-equivalent* states to get a "quotient structure" as defined below. We emphasize that any subgroup G of $Aut\ \mathcal{M} \cap Auto\ f$ is sufficient for our application. The largest one possible is desirable for maximal compression.

Let $\mathcal{M} = (\mathcal{S}, \mathcal{R})$ be a structure and let $\equiv$ be an equivalence relation on $\mathcal{S}$. Let $\overline{\mathcal{S}}$ be a *set of representatives* of the partition of $\mathcal{S}$ into equivalence classes induced by $\equiv$,i.e. for each $s \in \mathcal{S}$ there exists a unique *representative* $\overline{s}$ of s such that $\overline{s} \in [s] \cap \overline{\mathcal{S}}$. Then the *quotient of* $\mathcal{M}$ *modulo* $\equiv$, as specified by the set of representatives $\overline{\mathcal{S}}$, is $\overline{\mathcal{M}} = \mathcal{M}/\equiv\ = (\overline{\mathcal{S}}, \overline{\mathcal{R}})$ where $\overline{s} \longrightarrow \overline{t} \in \overline{\mathcal{R}}$ iff there exists $s' \equiv \overline{s}$ and there exists $t' \equiv \overline{t}$ such that $s' \longrightarrow t' \in \mathcal{R}$. When $\equiv$ is $\equiv_G$, for some G, we denote $\mathcal{M}/\equiv_G$ by $\mathcal{M}/G$ or simply by $\overline{\mathcal{M}}$.

3 Group-theoretic approach

3.1 Model Checking CTL*

In this section we present the correspondence lemma and the results showing that model checking of CTL* formulas on the original structure can be reduced to that on the quotient structure. Let $\mathcal{M} = (\mathcal{S}, \mathcal{R})$ and $\mathcal{M}/G = (\overline{\mathcal{S}}, \overline{\mathcal{R}})$ be as defined above. For a sequence $x = (s_0, s_1, ..., s_i, ...)$ of states in $\mathcal{S}$, we let $\overline{x}$ denote the sequence of corresponding representatives in $\overline{\mathcal{S}}$, i.e. $\overline{x} = (\overline{s_0}, \overline{s_1}, ..., \overline{s_i}, ...)$.

Lemma 3.1: (Correspondence Lemma) There is a bidirectional correspondence between paths of the original structure $\mathcal{M}$ and the quotient structure $\overline{\mathcal{M}} = \mathcal{M}/G$ for any $G \leq Aut\ \mathcal{M}$:
(i) If $x = s_0, s_1, s_2, \ldots$ is a path in $\mathcal{M}$, then $\overline{x}$ is a path in $\overline{\mathcal{M}}$.
(ii) If $\overline{x} = \overline{s}_0, \overline{s}_1, \overline{s}_2, \ldots$ is a path in $\overline{\mathcal{M}}$, then for every state $s'_0 \equiv_G \overline{s}_0$ in $\mathcal{M}$ there exists a corresponding path $x' = s'_0, s'_1, s'_2, \ldots$ in $\mathcal{M}$ of states such that $s'_i \equiv_G \overline{s}_i$.

Proof: Part (i) is immediate from the definition of quotient structure. To prove (ii), let $\overline{x} = \overline{s}_0, \overline{s}_1, \overline{s}_2, \ldots$ be a path in $\overline{\mathcal{M}}$. Choose an arbitrary $s'_0 \equiv_G \overline{s}_0$. By definition of quotient structure and since $\overline{s}_0 \longrightarrow \overline{s}_1 \in \overline{\mathcal{R}}$, there exists $s''_0 \equiv_G \overline{s}_0$ and there exists $s''_1 \equiv_G \overline{s}_1$ such that $s''_0 \longrightarrow s''_1 \in \mathcal{R}$.

Thus, by transitivity $s'_0 \equiv_G s''_0$ and $s'_0 = \pi(s''_0)$ for some permutation $\pi \in G$. Let $s'_1 = \pi(s''_1)$. Now, $s'_0 \longrightarrow s'_1 = \pi(s''_0) \longrightarrow \pi(s''_1) \in \mathcal{M}$ since $s''_0 \longrightarrow s''_1 \in \mathcal{M}$ and $\pi \in G \leq Aut\ \mathcal{M}$. Moreover, $s'_1 = \pi(s''_1) \equiv_G \overline{s}_1$ as desired.

The first edge of x' is thus defined by $s'_0 \longrightarrow s'_1$. Continuing with s'_1 the same argument can be applied to exhibit s'_2 such that $s'_1 \longrightarrow s'_2 \in \mathcal{M}$ and $s'_2 \equiv_G \overline{s}_2$. Proceeding, in this fashion we see that there is $s'_i \longrightarrow s'_{i+1} \in \mathcal{M}$ corresponding to each $\overline{s}_i \longrightarrow \overline{s}_{i+1}$ of $\overline{x}$ in $\overline{\mathcal{M}}$. The process continues for all natural numbers i or until the terminal i of $\overline{x}$ if it is finite. Let $x' = s'_0 \longrightarrow s'_1 \longrightarrow s'_2 \longrightarrow \ldots$ be the resulting path in $\mathcal{M}$. By construction, it corresponds to $\overline{x}$ in the desired way. □

Remark: If the Correspondence Lemma is restricted to paths consisting of a single transition, it amounts to saying that there is a bisimulation between $\mathcal{M}$ and $\overline{\mathcal{M}}$ defined by $\equiv_G$. □

Let f be any CTL* formula. We define a subset of subformulas of f, called *significant* subformulas, as follows.

- f is a significant subformula of itself.
- For every subformula g of f which is of the form $\mathsf{X}g'$ or $\mathsf{E}g'$, both g and g' are significant subformulas of f. Similarly, for every subformula g of the form $g'\ \mathsf{U}\ h'$, all of g, g' and h' are significant subformulas of f.

Intuitively, g is a significant subformula of f if either g is same as f, or the outer most connective of g is a temporal operator or is a path quantifier, or g appears as an immediate argument of a subformula whose outer most connective is a temporal operator or a path quantifier. For example, for the formula $f = \mathsf{E}g$ where g is given by $(P_1 \vee P_2) \wedge ((Q_1 \vee Q_2)\ \mathsf{U}\ (R_1 \vee R_2))$, the subformulas f, g, $(Q_1 \vee Q_2)\ \mathsf{U}\ (R_1 \vee R_2)$, $(Q_1 \vee Q_2)$, $(R_1 \vee R_2)$ and $(P_1 \vee P_2)$ are all the significant subformulas. Note that, in this case, none of the atomic propositions is a significant subformula.

Lemma 3.2: For every significant subformula h of f, $Auto\ f\ \subseteq\ Auto\ h$.

Proof: Let g be any significant subformula of f. We define the *immediate* significant subformulas of g as follows. If g is of the form $\mathsf{X}g'$ or $\mathsf{E}g'$ then g' is an immediate significant subformula of g. If g is of the form $g' \ \mathsf{U}\ h'$, then both g' and h' are the immediate significant subformulas of g. If neither of these conditions holds then g is a boolean expression over atomic propositions and significant subformula $f_1, ..., f_k$ where each f_i is of the form $\mathsf{X}g'$ or $\mathsf{E}g'$ or $g' \ \mathsf{U}\ h'$; in this case $f_1, ..., f_k$ are the immediate significant subformulas of g. From the definition of *Auto* g the following condition holds: for every immediate subformula g' of g, $Auto\ g \subseteq Auto\ g'$. Applying this inductively we get the following: for every significant subformula g' of g, $Auto\ g \subseteq Auto\ g'$. The lemma follows by using f and h in place of g and g', respectively, in the above observation. □

The Correspondence Lemma and the previous lemma make it easy to prove the following fundamental result showing that model checking over $\mathcal{M}$ can be reduced to model checking over $\overline{\mathcal{M}}$.

Theorem 3.3: (Compression Theorem for CTL*) For all structures $\mathcal{M}$, all CTL* formulas f, all subgroups $G \leq Aut\ \mathcal{M} \cap Auto\ f$, and all fullpaths x in $\mathcal{M}$,

$$\mathcal{M}, x \models f \quad \text{iff} \quad \mathcal{M}/G, \overline{x} \models f$$

Proof: We argue by induction on formula structure that, for every significant subformula g of f and every fullpath x in $\mathcal{M}$, $\mathcal{M}, x \models g$ iff $\mathcal{M}/G, \overline{x} \models g$. Formally, let $count(g)$ denote the number of occurrences in g of symbols from $\{\mathsf{U}, \mathsf{X}, \mathsf{E}\}$. We proceed by induction on $count(g)$, letting $x = (s_0, s_1, ..., s_i, ...)$.

Base Case: $count(g) = 0$. In this case g is a propositional formula. Hence, $\mathcal{M}, x \models g$ iff s_0 satisfies g and $\overline{\mathcal{M}}, \overline{x} \models g$ iff $\overline{s_0}$ satisfies g. Clearly, there exists a permutation $\pi \in G$ such that $\overline{s_0} = \pi(s_0)$. Using lemma 3.2 and the fact that g is a maximal subformula of f and $\pi \in Auto\ f$, we see that $\pi \in Auto\ g$. From this, we deduce that g is equivalent to $\pi(g)$. Clearly, s_0 satisfies g iff $\pi(s_0)$ satisfies $\pi(g)$ iff $\overline{s_0}$ satisfies g. Hence, $\mathcal{M}, x \models g$ iff $\mathcal{M}/G, \overline{x} \models g$.

Induction Step: Assume that the lemma holds for all maximal subformulas g' such that $count(g') \leq k$ and for all maximal paths in $\mathcal{M}$. Let g be any maximal subformula with $count(g) = k + 1$. Now, we have the following cases.

$g = \mathsf{E}g'$: $\mathcal{M}, x \models g$ iff there exists a maximal path x' in $\mathcal{M}$ starting from s_0 such that $\mathcal{M}, x' \models g'$. From the induction hypothesis and the fact that $count(g') = k$, it follows that $\mathcal{M}, x' \models g'$ iff $\mathcal{M}/G, \overline{x'} \models g'$. Since, $\overline{x'}$ and $\overline{x}$ start from the same state in $\overline{\mathcal{S}}$, it is the case that $\mathcal{M}/G, \overline{x'} \models g$ iff $\mathcal{M}/G, \overline{x} \models g$. The induction step follows from these observations.

$g = \mathsf{X}g'$: $\mathcal{M}, x \models g$ iff $\mathcal{M}, x^{(1)} \models g'$ iff $\mathcal{M}/G, \overline{x^{(1)}} \models g'$ iff $\mathcal{M}/G, \overline{x} \models g$. The second step follows from the induction hypothesis, and the last step follows from the fact that $\overline{x^{(1)}} = \overline{x}^{(1)}$ and the definition of nexttime.

$g = g' \ \mathsf{U}\ h'$: $count(g'), count(h') \leq k$. Now, $\mathcal{M}, x \models g$ iff for some $i \geq 0$, $\mathcal{M}, x^{(i)} \models h'$ and for all j, $0 \leq j < i$, $\mathcal{M}, x^{(j)} \models g'$. By induction hypothesis, the later condition holds iff $\mathcal{M}/G, \overline{x^{(i)}} \models h'$ and for all j, $0 \leq j < i$, $\mathcal{M}/G, \overline{x^{(j)}} \models g'$. The last condition holds iff $\mathcal{M}/G, \overline{x} \models g$. The induction step follows from these observations.

Other cases: In this case, it is easy to see that g is a boolean combination of atomic propositions and maximal subformulas whose outer most connective is from $\{\mathsf{X}, \mathsf{U}, \mathsf{E}\}$. That is, $g = b(e_1, e_2, ..., e_k,\ f_1, f_2, ..., f_l)$ where b is a boolean expression over the indexed atomic propositions $e_1, ..., e_k$ and maximal subformulas $f_1, ..., f_l$ such that, for each i, $1 \leq i \leq l$, the outer most connective of f_i belongs to $\{\mathsf{X}, \mathsf{U}, \mathsf{E}\}$ and $count(f_i) = k + 1$. Using the induction hypothesis for the previous three cases, we see that, for each i, $1 \leq i \leq l$, $\mathcal{M}, x \models f_i$ iff $\mathcal{M}/G, \overline{x} \models f_i$.

Now, let $F_1, ..., F_l$ be some distinct unindexed atomic propositions. From lemma 4.2, we get $Auto\ f \subseteq Auto\ g$. From the definition of $Auto\ g$, we get $Auto\ g \leq Auto\ b(e_1, ..., e_k, F_1, ..., F_l)$. Hence, $G \leq Auto\ b(e_1, ..., e_k, F_1, ..., F_l)$. Now, for each i, $1 \leq i \leq l$, let u_i be a boolean constant defined as follows: if $\mathcal{M}, x \models f_i$ then $u_i = true$ else $u_i = false$. It should be easy to see that $\mathcal{M}, x \models g$ iff the state s_0 satisfies $b(e_1, ..., e_k, u_1, ..., u_l)$ and $\mathcal{M}/G, \overline{x} \models g$ iff the state $\overline{s_0}$ satisfies $b(e_1, ..., e_k, u_1, ..., u_l)$. Let $\pi \in G$ be such that $\overline{s_0} = \pi(s_0)$. Clearly, s_0 satisfies $b(e_1, ..., e_k, u_1, ..., u_l)$ iff $\overline{s_0}$ satisfies $\pi(b(e_1, ..., e_k, u_1, ..., u_l))$. Since $G \leq Auto\ b(e_1, .., e_k, F_1, ..., F_l)$, it is the case that $\pi \in Auto\ b(e_1, ..., e_k, F_1, ..., F_l)$. From this, we see that $\pi(b(e_1, ..., e_k, u_1, ..., u_l))$ is equivalent to $b(e_1, ..., e_k, u_1, ..., u_l)$. Hence, s_0 satisfies $b(e_1, ..., e_k, u_1, ..., u_l)$ iff $\overline{s_0}$ satisfies $b(e_1, ..., e_k, u_1, ..., u_l)$. Putting all the above observations together, we get the induction step. □

3.2 Model checking the Mu-calculus

We now prove a result analogous to Theorem 3.3 showing that model checking for the Mu-calculus over the original structure $\mathcal{M}$ can be reduced to model checking over the quotient structure $\overline{\mathcal{M}}$.

First, we define the *significant* subformulas of a formula f as follows:

- f is a significant subformula of itself.
- Every variable appearing in f is a significant subformula.
- For every subformula g of f which is of the form $\mu y.g'$ or $\texttt{<R>}g'$, both g and g' are significant subformulas.

The following technical lemma is similar to lemma 3.2, and its proof is left to the reader.

Lemma 3.5: If h is a significant subformula of f then $Auto\ h \subseteq Auto\ f$.

Theorem 3.6: For every structure $\mathcal{M} = (\mathcal{S}, \mathcal{R})$, closed Mu-calculus formula f, subgroup G such that $G \leq Auto\ \mathcal{M} \cap Auto\ f$, and state s in $\mathcal{S}$, we have $\mathcal{M}, s \models f$ iff $\mathcal{M}/G, \overline{s} \models f$.

Let $\mathcal{M}$, f and G be as given in the statement of theorem 3.5. In order to prove the theorem, we need the following definitions. Let D, D' be subsets of $\mathcal{S}$ and $\overline{\mathcal{S}}$, respectively. We say that D and D' *correspond* if $D = \{s : \text{for some } t \in D'\ s \equiv_G t\}$; i.e., D is the union of all the equivalence classes that have a representative in D'. Let ρ and ρ' be evaluations having same domains and with ranges being the power sets of $\mathcal{S}$ and $\overline{\mathcal{S}}$, respectively. We say that ρ and ρ' *correspond* if for each variable x, $\rho(x)$ and $\rho'(x)$ correspond.

Proof of Theorem 3.6: We argue by induction on formula structure that, for every significant subformula g of f and for any two evaluations ρ and ρ' that correspond with each other, $\mathcal{L}_{(\mathcal{M},g)}(\rho)$ and $\mathcal{L}_{(\overline{\mathcal{M}},g)}(\rho')$ correspond with each other. Intuitively, this asserts that for any significant subformula g of f, a state s in the structure $\mathcal{M}$ satisfies g with respect to an evaluation ρ iff its representative $\overline{s}$ in $\overline{\mathcal{M}}$ satisfies g with respect to a corresponding evaluation ρ'. Formally, for any significant subformula g, let $count(g)$ denote the number of occurrences in g of symbols from the set $\{\mu, \texttt{<R>}\}$. We proceed by induction on $count(g)$.

Base Case: $count(g) = 0$. In this case, g is simply a propositional formula over atomic propositions and variables. From lemma 3.6, it is the case that $Auto\ f \subseteq Auto\ g$ and hence $G \leq Auto\ g$. ¿From these observations it should be easy to see that $\mathcal{L}_{\mathcal{M},g}(\rho)$ and $\mathcal{L}_{\overline{\mathcal{M}},g}(\rho')$ correspond.

Induction Step: Assume that the theorem holds for all significant subformulas g' such that $count(g') \leq k$. Let g be a significant subformula such that $count(g) = k+1$. Now we have the following cases.

g = `<R>`g': This case is straightforward from the induction hypothesis.

$g = \mu y.g'$: Let $\rho_0, \rho_1, ..., \rho_l$ and $\rho'_0, \rho'_1, ..., \rho'_l$ be sequences of evaluations obtained by iteratively computing the least fixed points in the structures $\mathcal{M}$ and $\mathcal{M}/G$, respectively. Formally, these sequences of evaluations are defined as follows. For $i = 0, ..., l$ and for each $z \neq y$, $\rho_i(z) = \rho(z)$ and $\rho'_i(z) = \rho'(z)$. $\rho_0(y) = \rho'_0(y) = \emptyset$. For $i = 1, ...l$, $\rho_i(y) = \mathcal{L}_{\mathcal{M},g'}(\rho_{i-1})$ and $\rho'_i(y) = \mathcal{L}_{\overline{\mathcal{M}},g'}(\rho'_{i-1})$; $\rho_l = \rho_{l-1}$ and $\rho'_l = \rho'_{l-1}$. Using the main induction hypothesis and by induction on i, it can easily be shown that ρ_i and ρ'_i correspond for $i = 0, ..., l$. By Tarski-Knaster theorem, it is the case that $\mathcal{L}_{\mathcal{M},g}(\rho) = \rho_l(y)$ and $\mathcal{L}_{\overline{\mathcal{M}},g}(\rho') = \rho'_l(y)$. Hence $\mathcal{L}_{\mathcal{M},g}(\rho)$ and $\mathcal{L}_{\overline{\mathcal{M}},g}(\rho')$ correspond.

Other Cases: In this case g is a boolean combination of atomic propositions, variables and significant subformulas of the form `<R>`g' or $\mu y.g'$. Hence $g = b(e_1, ..., e_l, f_1, ..., f_m)$ where each e_i is an atomic proposition or a variable, and each f_i is a formula of the form `<R>`g' or $\mu y.g'$ such that $count(f_i) \leq k+1$. Using the induction step for the previous two cases, we can show that $\mathcal{L}_{\mathcal{M},f_i}(\rho)$ and $\mathcal{L}_{\overline{\mathcal{M}},f_i}(\rho')$ correspond. ¿From this observation and the fact that $G \leq$ *Auto* $f \subseteq$ *Auto* $b(e_1, ..., e_l, F_1, ..., F_m)$, where $F_1, ..., F_m$ are unindexed atomic propositions, it is easy to show that $\mathcal{L}_{\mathcal{M},g}(\rho)$ and $\mathcal{L}_{\overline{\mathcal{M}},g}(\rho')$ correspond.

The theorem now follows by taking f itself for g and the empty evaluation, assigning $false$ to every variable, for ρ and ρ'. □

4 Applications

We wish to determine whether $\mathcal{M}, s_0 \models f$, where $\mathcal{M}$ is the global state transition graph of $\mathcal{P} = //_i K_i$ and f is an arbitrary CTL* or Mu-calculus formula, without incurring the potentially enormous cost of constructing $\mathcal{M}$. By theorem 3.3, it suffices to construct $\mathcal{M}/G$, where G is a subgroup of *Aut* $\mathcal{M} \cap$ *Auto* f, and then check whether $\mathcal{M}/G, s_0 \models f$. If G is large, reflecting a good deal of symmetry common to $\mathcal{M}$ and f, then we should realize a significant savings.

4.1 Determination of a Suitable Group G

We can take G to be $G' \cap$ *Auto* f for any subgroup G' of *Aut* $\mathcal{M}$. Thus, to calculate G we should determine (i) *Auto* f, (ii) largest possible G' and (iii) the intersection of (i) and (ii). Each of these appears to be a difficult problem. Fortunately, with certain reasonable restrictions on $\mathcal{M}$ and f the computations of (i)-(iii) become much easier.

It is to be noted that if we have an algorithm for computing *Aut* p for a propositional formula p, then we can use it inductively to compute *Auto* f for an arbitrary CTL* formula f. However, automatic computation of *Aut* p for a propositional formula p is a computationally hard problem. For example, the following proposition shows that three important problems associated with *Aut* p, namely *universality*, *membership* and *non-triviality* are computationally hard problems. Here the universality problem is to determine if *Aut* p = *Sym* I for a propositional formula p. The membership problem is to check if a given permutation π is in *Aut* p for a propositional formula p. The *non-triviality* problem is to check that there exists at least one non-identity permutation in *Aut* p.

Proposition 4.1: All the three problems, i.e. universality, membership and non-triviality, are co-NP-hard.

Proof. We reduce the validity problem for propositional formulas to the universality problem. Let p be any propositional formula. It has some number n of atomic propositions, and we may assume without loss of generality, that they are indexed by $I = [1 : n]$, viz. $Q_1, \ldots, Q_n$. We may also assume that there exists an alphabet of n distinct propositions P_i, also indexed by I, such that no P_i appears in p.

We claim that p is valid iff $Aut\ (p \vee P_1)\ =\ Sym\ I$. If p is valid, then any formula resulting from any permutation of the indices on its propositions is also valid, and similarly for the validity $p \vee P_1$; hence, $Aut\ (p \vee P_1)\ =\ Sym\ I$. Conversely, let us assume that $Aut\ (p \vee P_1)\ =\ Sym\ I$. Let π be a permutation such that $\pi(1)\ =\ 2$. Now, by assumption, we have that $\pi(p \vee P_1)\ \equiv\ (p \vee P_1)$, simplifying to $\pi(p) \vee P_2\ \equiv\ p \vee P_1$. Consider any assignment of truth values to all the propositions Q_i. Extend it to assignment to all the propositions such that P_1 and P_2 are set to *false* and *true*, respectively; for any such assignment p should evaluate to *true*. Since no P_i, for any i, appears in p, we can conclude that p is valid.

With slight modifications we can also exhibit similar reductions to the other two problems $\square$

It should be obvious that both the universality and membership problem are in co-NP, and hence are co-NP-complete.

In practice, for a CTL* formula f, $Auto\ f$ can often be determined through inspection or by using some heuristics. For example, if $f\ =\ \mathsf{AF}(C_1 \vee C_2 \vee \ldots \vee C_n)$ then it is easy to see that $Auto\ f\ =\ Sym\ I$.

For many systems we can also determine $Aut\ \mathcal{M}$ by inspection of program $\mathcal{P}$, and in these cases we can take G to be $Aut\ \mathcal{M} \cap Auto\ f$. Sometimes, we can take G to be $Aut\ \mathcal{P}\ \cap\ Auto\ f$. Here $Aut\ \mathcal{P}$ is the set of automorphisms of the program $\mathcal{P}$ defined as follows.

In order to define $Aut\ \mathcal{P}$, we first define the notion of equivalences of transitions and equivalence of processes. Recall that each process is comprised of a set of transitions. We say that a transition $l : B \rightarrow A : m$ is *equivalent* to another transition $l' : B' \rightarrow A' : m'$ if $l = l', m = m'$, the boolean expressions B, B' are (semantically) equivalent, and finally A, A' are (semantically) equivalent, i.e. update the same variables and assign equivalent expressions to identical variables. We say that two processes K and K' are equivalent if there exists a bijection mapping each transition of K to an *equivalent* transition of K'.

Now, let $\mathcal{P}\ =\ //_i K_i$ be a program (with start state s_0). Let π be a permutation on the process indices. We extend π to the processes in $\mathcal{P}$ as follows. For each K_i, let $\pi(K_i)$ be the process obtained by replacing each occurrence of index j by $\pi(j)$ for each $j \in I$. We say that a permutation π on process indices is an *automorphism* of $\mathcal{P}$ if for each i, $\pi(K_i)$ and $K_{\pi(i)}$ are equivalent (and $\pi(s_0) = s_0$). Let $Aut\ \mathcal{P}$ denote the set of automorphisms of $\mathcal{P}$. Clearly, $Aut\ \mathcal{P}$ forms a group.

Lemma 4.2: $Aut\ \mathcal{P} \ \leq\ Aut\ \mathcal{M}$ where $\mathcal{M}$ is the global transition graph of program $\mathcal{P}$ (with start state s_0).

Proof: Let π be a permutation in $Aut\ \mathcal{P}$. Assume $s \longrightarrow t \in \mathcal{M}$. Then for some i, $s \longrightarrow_i t$, and in process K_i there is some local transition τ driving s to t in $\mathcal{M}$. Clearly, the transition $\pi(\tau)$ in $\pi(K_i)$ drives $\pi(s)$ to $\pi(t)$ in $\pi(\mathcal{M})$. Since the processes $\pi(K_i)$ and $K_{\pi(i)}$ are equivalent, there is a transition τ' in $K_{\pi(i)}$ of the program $\mathcal{P}$ generating $\mathcal{M}$ that is (semantically) equivalent to $\pi(\tau)$ and that drives $\pi(s)$ to $\pi(t)$ in $\mathcal{M}$. Hence, $\pi(s) \longrightarrow_{\pi(i)} \pi(t)$, and $\pi(s) \longrightarrow \pi(t)\ \in \mathcal{M}$. Since the above property holds for any transition $s \longrightarrow t$ of $\mathcal{M}$, we conclude that $\pi\ \in\ Aut\ \mathcal{M}$, while noting that if $\mathcal{P}$ has start state s_0, it must be that $\pi(s_0) = s_0$. $\square$

It may not be easy to determine the automorphism group of $\mathcal{P}$. However, sometimes when all the processes in $\mathcal{P}$ are *normal* and are *isomorphic*, we can use the automorphism group of the underlying communication graph to determine an appropriate G. We formalize this below.

Let $\mathcal{P} = //_i K_i$ be the concurrent program. We assume that each shared variable in $\mathcal{P}$ is shared between exactly two processes. Corresponding to $\mathcal{P}$, we define an undirected graph CR as follows. The nodes of CR are the process indices and there is an edge connecting i and j iff i and j share a variable x_{ij}. (Shared variable x_{ij} is also equivalently denoted x_{ji}.) For any node i, we let $CR(i)$ denote the neighbors of i. We say that the processes in $\mathcal{P}$ are *normal* if every transition τ in each process K_i is of the form:

$$\ell : \bigwedge_{j \in CR(i)} B(i,j) \rightarrow \|_{j \in CR(i)}\ A(i,j) : m$$

where $B(i,j)$ is a boolean expression over atomic formulas that are either atomic propositions Q_i or Q_j, or equality tests of shared variables of the form $x_{ij} = y_{ij}$ or $x_{ij} = d$, where x, y are variable names and d is the name of a domain element; and $A(i,j)$ is a concurrent assignment to variables shared between i and j of the form $x_{ij} := y_{ij}$ or $x_{ij} := d$.

We say that two processes K_i and $K_{i'}$, where $i \equiv_{Aut\ CR} i'$, are *isomorphic* if there exists a bijection mapping each transition $\tau \in K_i$ to a transition $\tau' \in K_{i'}$ such that if τ is

$$\ell : \bigwedge_{j \in CR(i)} B(i,j) \rightarrow \|_{j \in CR(i)}\ A(i,j) : m$$

then τ' is

$$\ell : \bigwedge_{j \in CR(i')} B(i',j) \rightarrow \|_{j \in CR(i')}\ A(i',j) : m.$$

It should be noted that $B(i',j)$ is the same $B(i,j)$ and $A(i',j)$ is the same as $A(i,j)$ except that the subscript i is replaced by i'.

Theorem 4.3: If $\mathcal{M}$ is the global state transition graph of $\mathcal{P} = //_i K_i$ where all K_i are normal and isomorphic then $Aut\ CR\ \leq\ Aut\ \mathcal{M}$.

Proof: We will show that $Aut\ CR\ \ \leq\ \ Aut\ \mathcal{P}$, and from lemma 4.2, it would follow that $Aut\ CR\ \leq\ Aut\ \mathcal{M}$. Let $\pi\ \in\ Aut\ CR$. For any i, consider the processes K_i and $K_{\pi(i)}$. Since these two processes are normal and isomorphic, there exists a bijection that maps each transition τ of K_i of the form

$$\ell : \bigwedge_{j \in CR(i)} B(i,j) \rightarrow \|_{j \in CR(i)}\ A(i,j) : m$$

to a transition τ' of $K_{\pi(i)}$ which is of the form

$$\ell : \bigwedge_{j \in CR(\pi(i))} B(\pi(i), j) \to \|_{j \in CR(\pi(i))} A(\pi(i), j) : m.$$

Since $\pi \in Aut\ CR$, the transition $\pi(\tau)$ is equivalent to τ'. Hence the processes $\pi(K_i)$ and $K_{\pi(i)}$ are equivalent, and $\pi \in Aut\ \mathcal{P}$. □

Since in designing the program $\mathcal{P}$, the choice of CR is (one hopes!) explicitly and carefully considered, and often chosen from a standard pattern, determination of $Aut\ CR$ is often easy in practice, and frequently is just a well-known fact of graph theory. We have for example:

- If $CR = I \times I \setminus Id$ so that the communication topology is the complete graph on I, then $Aut\ CR = Sym\ I$.
- If the processes $K_1, \ldots, K_n$ of $\mathcal{P}$ are arranged in a ring then $CR = \{(i, i \oplus_n 1), (i, i \ominus n1) : i \in I\}$, where $\oplus_n$ denotes wrap-around addition where $n \oplus_n 1 = 1$, and analogously for subtraction. This indicates that each process can only communicate with its two neighbors in the ring. Thus, $Aut\ CR = D_n$, the dihedral group of order $2n$.

To determine the intersection of $Auto\ f$ and $Aut\ CR$, we can often proceed by inspection. In practice, it is likely to turn out that one or both of $Auto\ f$ or $Aut\ CR$ is large, for example $Sym\ I$ or $Sym\ I \setminus \{i\}$, or at least a well-known permutation group which simplifies our task.

4.2 Constructing the Quotient Structure

We can construct $\mathcal{M}/G$ from $\mathcal{P}$ with start state s_0 incrementally, without building $\mathcal{M}$ itself, as shown in figure 1 (cf. [ID92], [LY92], [CFJ93]).

An important part of the above procedure is the test $t \equiv_G \overline{u}$. Since G may in the worst case be $Aut\ \mathcal{M}$, this could conceivably be intractable (cf. [Ho82], [CFJ93]). However, in practice $\mathcal{M}$ has special structure derived from $\mathcal{P}$, which can simplify matters. In some cases, the test is particularly simple. For example, if $\mathcal{S} = L^I$ and $L = \{\ell_1, \ldots, \ell_m\}$ and $G = Sym\ I$, then $s \equiv_G t$ iff for each $i \in [1 : m]$ the number of processes in local state ℓ_i is the same for both global states s and t.

4.3 Decomposing Formulae

In some instances G may be very small essentially because f is a large composite formula. Consider, for example, $f = \wedge_i \mathsf{AG}(N_i \Rightarrow \mathsf{AF}C_i)$. We see that $Auto\ f = Pstab\ 1\ \cap \ldots \cap\ Pstab\ n = Id$. Since $G \leq Auto\ f$, it is the case that $G = Id$. So, no compression is possible in forming the quotient $\overline{\mathcal{M}} = \mathcal{M}/G$. Sometimes it is possible to overcome this problem by breaking down the composite formula into its basic modalities (or other appropriate subformulae) and checking them individually. While this may entail computing multiple quotients, it can still be more efficient. For the formula f specified above, we can check for each conjunct $f_i = \mathsf{AG}(N_i \Rightarrow \mathsf{AF}C_i)$ in turn. Since $Auto\ f_i = Pstab\ i = Sym\ I \setminus \{i\}$ is of exponential size, any G obtained from such an $Auto\ f_i$ is likely to be large. Thus computing n different exponentially smaller quotients can be more efficient than computing one large quotient, actually equal to the full, original structure.

4.4 State Symmetry

Sometimes we can take advantage of symmetry in the initial states to achieve faster model checking. Suppose s is a state that is fully symmetric in a fully symmetric structure $\mathcal{M}$, viz. $Aut\ s = Aut\ \mathcal{M} = Sym\ I$. For example, s could be the start state $(N_1, \ldots, N_n)$ for a solution

```
Let S̄ := ∅
Let s̄0 := s0
Add s̄0 to S̄
While unprocessed(S̄) ≠ do
    Remove some unprocessed s̄ from S̄
        For each i ∈ [1 : n] do
            For each t ∈ K_i(s̄) do
                Ensure t̄ ends up in S̄:
                    If ∃ū ∈ S̄ t ≡_G ū then
                        Note t̄ = ū ∈ S̄ already
                    Else
                        Let t̄ := t
                        Add t̄ to unprocessed(S̄)
                Add s̄ ⟶ t̄ to R̄
            End
        End
    Mark s̄ processed
End
```

Figure 1: Incremental Construction of Quotient

to the mutual exclusion problem with each process in its noncritical region (cf. [AE89], [EC82], section 6).

Consider the formula $\wedge_i g_i$ where g_i is a temporal formula over the atomic propositions with index i. Here $\wedge_i$ denotes a conjunction over all process indices i, i.e. all $i \in I$. Now, it can be shown that $\mathcal{M}, s \models \wedge_i\, g_i$ iff $\mathcal{M}, s \models g_1$. The $\Rightarrow$ direction is obvious. To see the $\Leftarrow$ direction, choose an arbitrary $i \in I$. Then pick some $\pi \in Aut\ s = Sym\ I$ such that $\pi(1) = i$. The right-hand-side implies that for all permutations π' that $\mathcal{M}, \pi'(s) \models \pi'(g_1)$ and hence $\mathcal{M}, \pi'(s) \models g_{\pi'(1)}$; this is due to the fact that each permutation π' is in $Aut\ \mathcal{M}$. For $\pi' = \pi$ this simplifies to the desired property $\mathcal{M}, s \models g_i$.

Thus, in reference to the previous section 4.3, in checking a formula such as $\wedge_i \mathsf{AG}(N_i \Rightarrow \mathsf{AF}C_i)$ evaluation of multiple conjuncts over multiple quotients is not required if the initial state and the structure are fully symmetric.

This idea can be generalized to states and systems with somewhat less symmetry. Let s be any state in $\mathcal{M}$. $Aut\ s \cap Aut\ \mathcal{M}$ induces an equivalence relation on I: $i \equiv j$ iff $i = \pi(j)$ for some $\pi \in Aut\ s \cap Aut\ \mathcal{M}$. Let $Part$ be the partition induced by the above equivalence relation. Let Rep be a set of representatives, one from each equivalence class in $Part$.

Theorem 4.4: $\mathcal{M}, s \models \wedge_i\, g_i$ iff $\mathcal{M}, s \models \wedge_{j \in\, Rep}\, g_j$

Proof: The $\Rightarrow$ direction is obvious. To see the $\Leftarrow$ direction, assume the left hand side holds. Choose an arbitrary $i \in I$. Let j be the representative equivalent to i. For some $\pi \in Aut\ s \cap Aut\ \mathcal{M}$, we have $i = \pi(j)$. Moreover, $\mathcal{M}, s \models g_j$. So $\mathcal{M}, \pi(s) \models \pi(g_j)$ because $\pi \in Aut\ \mathcal{M}$. Because $\pi(s) = s$, $\pi(g_j) = g_{\pi(j)}$ and $\pi(j) = i$, the above simplifies to $\mathcal{M}, s \models g_i$. □

Thus, instead of checking all $n = |I|$ conjuncts, it suffices to check $|Rep|$ conjuncts which may be significantly smaller. In the extreme case, as above, only one conjunct need be checked. If $Aut\ \mathcal{M} = Sym\ I$ then matters simplify so that at most $|L|$, the number of distinct local states, need be checked. Typically $|L| \ll n = |I|$. If $Aut\ s = Sym\ I$, so that s is a start state with all process in the same local state, then if $Aut\ CR$ is nontrivial, some equivalence class on I has 2 or more members, $|Rep| < n$, and some savings is obtained. In many practical cases $Aut\ CR$ may yield a small $|Rep|$. Any of the vertex-transitive connectivity graphs, which includes such "sparse" graphs as rings, yields only a single equivalence class.

5 Automata-theoretic Approach

We can give an alternative, uniform method using automata for model checking temporal properties of systems of processes that exhibit symmetry. The main feature of this approach is that a single, annotated quotient structure $\overline{\mathcal{M}} = \mathcal{M}/G$, where G is a subgroup of $Aut\ \mathcal{M}$, can be used to model check with respect to a variety of different specifications f. Each transition in the annotated quotient structure is labelled with additional information denoting how coordinates are permuted from one state to the next state. The annotated quotient structure is a succinct representation of the original structure. In order to verify that all computations satisfy a linear temporal specification f, we construct an automaton $\mathcal{A}$ that accepts exactly those strings that satisfy the formula $\neg f$, construct the cross product of $\overline{\mathcal{M}}$ with $\mathcal{A}$ and check that the product automaton does not accept any input strings.

5.1 The Annotated Quotient Structure

Let $\mathcal{M}$ =$(\mathcal{S}, \mathcal{R})$ be a structure which, for ease of exposition, is assumed to be total. We first fix a subgroup G of $Aut\ \mathcal{M}$. We then define the *annotated quotient structure* of $\mathcal{M}$ with respect to G, denoted $\overline{\mathcal{M}}$, to be $(\overline{\mathcal{S}}, \mathcal{AR})$ where $\overline{\mathcal{S}}$ is the set of representative states as before, and $\mathcal{AR}$ is an annotated relation consisting of the following elements. Corresponding to each transition $(\overline{s}, t) \in \mathcal{R}$, the triple $(\overline{s}, \pi, \overline{t})$, where $t = \pi(\overline{t})$ for some $\pi \in G$, is contained in $\mathcal{AR}$. All transitions in the original structure from a representative state $\overline{s}$ to another representative state $\overline{t}$ are included in $\mathcal{AR}$ as triples $(\overline{s}, \pi, \overline{t})$ in which the permutation π is the identity permutation. Also, all transitions in the original structure from a representative state $\overline{s}$ to a non-representative state t are encoded by some $(\overline{s}, \pi, \overline{t})$ in $\mathcal{AR}$ where π is not the identity. Those transitions from a non-representative state to a non-representative state in the original structure are not included in $\overline{\mathcal{M}}$. Due to this, many times, the size of the structure $\overline{\mathcal{M}}$ can be much smaller than that of $\mathcal{M}$. It is not difficult to see that we can obtain the original structure from the annotated quotient structure $\overline{\mathcal{M}}$.

We prove some simple properties of the annotated quotient structure $\overline{\mathcal{M}}$. An annotated path p in $\overline{\mathcal{M}}$ is an alternating infinite sequence $\overline{s_0}, \pi_1, \overline{s_1}, ..., \overline{s_i}, \pi_{i+1}, ...$ of states and permutations such that, for all $i \geq 0$, $(\overline{s_i}, \pi_{i+1}, \overline{s_{i+1}}) \in \mathcal{AR}$. We define a function h mapping annotated paths of $\overline{\mathcal{M}}$ to paths of $\mathcal{M}$ as follows. For any annotated path p as given above, $h(p) = t_0, t_1, ..., t_i, ...$ where $t_0 = \overline{s_0}$ and $t_i = \pi_1 \circ \pi_2 \circ \ldots \circ \pi_i(\overline{s_i})$ for all $i > 0$.

Lemma 5.1: The following properties are satisfied by $\overline{\mathcal{M}}$.

- For every annotated path p in $\overline{\mathcal{M}}$, $h(p)$ is a path in $\mathcal{M}$.
- For every path q in $\mathcal{M}$ starting from a representative state $\overline{s_0}$, there exists an annotated path p in $\overline{\mathcal{M}}$ such that $q = h(p)$.

Proof: To prove the first part of the lemma, assume that $p = t_0, \pi_1, t_1, \pi_2, ..., t_i, \pi_{i+1}, ...$ is an annotated path in $\overline{\mathcal{M}}$. ¿From the definition of $\overline{\mathcal{M}}$, it should be easy to see that, for each $i \geq 0$ the following properties are satisfied: $t_i \rightarrow \pi_{i+1}(t_{i+1})$ is a transition in $\mathcal{M}$. Since $\pi_0, \pi_1, \ldots$ are all in the group G of automorphisms of $\mathcal{M}$, it follows that that $\pi_1 \circ \pi_2 \circ ... \circ \pi_i(t_i) \rightarrow \pi_1 \circ \pi_2 \circ ... \circ \pi_{i+1}(t_{i+1})$ is a transition of $\mathcal{M}$. Hence $h(p)$ is a path in $\mathcal{M}$.

To prove the second part, we note that we can write any path $q = \overline{s_0}, s_1, s_2, \ldots$ in $\mathcal{M}$ starting at a representative state $\overline{s_0}$ in the form $\overline{s_0}, \phi_1(\overline{s_1}), \phi_2(\overline{s_2}), \ldots$ where, for each $j \geq 1$, ϕ_j is any permutation in G such that $\phi_j(\overline{s_i}) = s_i$. We will argue by induction on j that we can take ϕ_j to be a permutation of the form $\pi_1 \circ \ldots \circ \pi_j$ where $(\overline{s_0}, \pi_1, \overline{s_1}), \ldots, (\overline{s_j}, \pi_j, \overline{s_{j+1}}) \in \mathcal{AR}$. For $j = 1$, since $(\overline{s_0}, \phi_1(\overline{s_1})) \in \mathcal{R}$, by definition of $\overline{M}$, there is some $\pi_1 \in G$ and $(\overline{s_0}, \pi_1, \overline{s_1}) \in \mathcal{AR}$ such that

$\pi_1(\overline{s_1}) = \phi_1(\overline{s_1})$. Thus, we can take ϕ_1 to be π_1. Inductively, we can take $\phi_j = \pi_1 \circ \ldots \circ \pi_j$. Because $(\phi_j(\overline{s_j}), \phi_{j+1}(\overline{s_{j+1}})) \in \mathcal{R}$, we have $(\overline{s_j}, (\pi_1 \circ \ldots \circ \pi_j)^{-1} \circ \phi_{j+1}(\overline{s_{j+1}})) \in \mathcal{R}$, by induction hypothesis and since ϕ_j^{-1} is an automorphism of $\mathcal{M}$. Hence, there is some $\pi_{j+1} \in G$ and some $(\overline{s_j}, \pi_{j+1}, \overline{s_{j+1}}) \in \mathcal{AR}$ such that $\pi_{j+1}(\overline{s_{j+1}}) = (\pi_1 \circ \ldots \circ \pi_j)^{-1} \circ \phi_{j+1}(\overline{s_{j+1}})$. Thus, we can take $\phi_{j+1} = \pi_1 \circ \ldots \circ \pi_j \circ \pi_{j+1}$, thereby completing the induction step. Then, the annotated path $p = \overline{s_0}, \pi_1, \overline{s_1}, \pi_2, \overline{s_2} \ldots$ is such that $h(p) = q$. □

5.2 Model Checking Indexed CTL*

The above lemma allows us to model check properties specified in an Indexed CTL* (ICTL*) efficiently. The set of ICTL* formulas are defined inductively. To do this, we assume that the set $\mathcal{AP}$ of atomic propositions is partitioned into two sets $\mathcal{AP}'$ and $\mathcal{AP}''$ where $\mathcal{AP}'$ is the set of global propositions and $\mathcal{AP}''$ is the set of local propositions. We further assume that the set $\mathcal{AP}''$ is an indexed set, while $\mathcal{AP}'$ is not an indexed set. Global propositions denote global properties of a state, while local propositions denote properties of individual processes. An element $P_i \in \mathcal{AP}''$ indicates a property of process i, and its satisfaction in a global state depends only on the state of process i. We also assume that all the states in an equivalence class satisfy the same set of global propositions (these are same as the invariant propositions of [CFJ93]). The set of ICTL* formulas are defined inductively using the propositional connectives, atomic propositions and quantified formulas of the form $\vee_i \mathsf{E} f_i$ and $\wedge_i \mathsf{E} f_i$ where, f_i is any propositional linear temporal logic (PLTL) formula that only uses global propositions and local propositions of process i. The symbol $\vee_i$ acts as an existential quantifier ranging over processes indices. Similarly, $\wedge_i$ acts as a universal process quantifier. E acts as an existential path quantifier. We further stipulate that all local propositions should appear in the scope of a process quantifier.

Lemma 5.2: Two equivalent states in $\mathcal{M}$ satisfy the same set of ICTL* formulas.

Proof: Let s and t be two states such that $t = \pi(s)$ for some $\pi \in G$. For any ICTL* formula f, s satisfies f iff t satisfies $\pi(f)$. Roughly speaking, the above property is satisfied due to the fact that, the tree rooted at the state t in $\mathcal{M}$ is obtained by taking the tree rooted at s and replacing each state s' in the tree by the state $\pi(s')$. In addition, since f is an ICTL* formula, it is the case that f and $\pi(f)$ are equivalent. Hence s satisfies f iff t satisfies f. □

¿From the above lemma it is enough if we give a procedure to check if a representative state satisfies an ICTL* formula. Furthermore, it is enough if we give the procedure for ICTL* formulas of the form $\vee_i \mathsf{E} f_i$ and $\wedge_i \mathsf{E} f_i$. We show how to model check for these type of formulas using the annotated quotient structure.

As indicated previously, we will be using automata for model checking temporal properties. A Buchi automaton $\mathcal{A}$ on infinite strings is quintuple $(Q, \Sigma, \delta, I, R)$ where Q is a finite set of automaton states, Σ is the input alphabet, $\delta : (Q \times \Sigma) \rightarrow 2^Q$ is the transition function, $I \subseteq Q$ is the set of *initial* states and $R \subseteq Q$ is the set of *recurrent* states. A run of the automaton on an input $t = (t_0, ...t_i, ...) \in \Sigma^\omega$ is an infinite sequence $(q_0, ..., q_i, ...)$ of automaton states such that $q_0 \in I$, and for all $i \geq 0$, $q_{i+1} \in \delta(q_i, t_i)$. We say that a run is accepting iff some recurrent state occurs infinitely often in the run. We say that an input $t \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ iff there is an accepting run of $\mathcal{A}$ on t.

We first construct a Buchi automaton $\mathcal{A}$ corresponding to the PLTL formula f_i and check that there is no path in $\mathcal{M}$ that is accepted by it. The input alphabet of $\mathcal{A}$ is the set of subsets of local propositions and global propositions. We next construct a directed graph $\overline{\mathcal{B}}$ which is a product of the annotated structure and the automaton $\mathcal{A}$. The nodes of $\overline{\mathcal{B}}$ are triples of the form $(\overline{s}, q, j)$ where $\overline{s} \in \overline{\mathcal{S}}$, q is a state of the automaton $\mathcal{A}$ and j is a process index. The edges/transitions of $\overline{\mathcal{B}}$ are defined as follows. For every transition of the form $(\overline{s}, \pi, \overline{t}) \in \mathcal{AR}$ and for every automaton state q and process index j, there is going to be an edge in $\overline{\mathcal{B}}$ from node $(\overline{s}, q, j)$ to the node $(\overline{t}, r, \pi^{-1}(j))$ where r is any state to which there is a transition of $\mathcal{A}$ from state q on the input which is the set of global propositions satisfied in $\overline{s}$ and local propositions satisfied in the process j's component of $\overline{s}$. We say that a node $(\overline{s}, q, j)$ of $\overline{\mathcal{B}}$ is a *recurrent* node iff q is a recurrent state of $\mathcal{A}$. Let q_0 be the initial state of $\mathcal{A}$.

Lemma 5.3: The following properties hold for all $\overline{s} \in \overline{\mathcal{S}}$.

- The formula $\vee_i \mathsf{E} f_i$ is satisfied in the state $\overline{s}$ of the structure $\mathcal{M}$ iff for some i, $1 \leq i \leq n$, there exists an infinite path in $\overline{\mathcal{B}}$ starting from the node $(\overline{s}, q_0, i)$ and containing infinitely many recurrent nodes.

- The formula $\wedge_i \mathsf{E} f_i$ is satisfied in the state $\overline{s}$ of the structure $\mathcal{M}$ iff for all i, $1 \leq i \leq n$, there exists an infinite path in $\overline{\mathcal{B}}$ starting from the state $(\overline{s}, q_0, i)$ and containing infinitely many recurrent nodes.

Proof: We prove the first part of the lemma. The second part can be proved analogously. Assume that the formula $\vee_i \mathsf{E} f_i$ is satisfied in the state $\overline{s}$. Let $p = s_0, s_1, ..., s_j, ...$ be a path in $\mathcal{M}$ and i_0 be a process index such that $\overline{s} = s_0$ and p satisfies the formula f_{i_0}. Let $q_0, q_1, ..., q_j, ...$ be an accepting run of $\mathcal{A}$ on the above path. From lemma 5.1, we know that there exists an annotated path $p' = t_0, \pi_1, t_1, \pi_2, ...t_j, \pi_{j+1}, ...$ such that $h(p') = p$. Now define a sequence of process indices $i_0, i_1, ..., i_j, ...$ such that $i_j = \pi_j^{-1} \circ \pi_{j-1}^{-1} \circ ... \circ \pi_1^{-1}(i_0)$. From the definition of $\overline{\mathcal{B}}$, it can be shown that the sequence $(t_0, q_0, i_0), (t_1, q_1, i_1), ...(t_j, q_j, i_j)...$ is a path in $\overline{\mathcal{B}}$. This path contains infinitely many recurrent nodes, and in addition $t_0 = \overline{s}$.

To prove the other direction of the first part, let $(s_0, q_0, i_0), (s_1, q_1, i_1), ..., (s_j, q_j, i_j), ...$ be a path in $\overline{\mathcal{B}}$ that contains infinitely many recurrent nodes and such that $s_0 = \overline{s}$. From the construction of $\overline{\mathcal{B}}$ we see that there exists an annotated path $p' = s_0, \pi_1, s_1, \pi_2, ..., s_j, \pi_{j+1}, ...$ in $\overline{\mathcal{M}}$ such that, for each $j > 0$ $i_j = \pi_j^{-1}(i_{j-1})$, and q_j is such that there is a transition of $\mathcal{A}$ from the state q_{j-1} on the input which is the set of global propositions and local propositions satisfied by process i_{j-1} in the state s_{j-1}. Let $h(p) = t_0, t_1, ..., t_j, ...$ where $t_j = \pi_1 \circ \pi_2 \circ ...\pi_j(s_j)$. From lemma 5.1, we see that $h(p)$ is a path in $\mathcal{M}$. Also, it is not difficult to see that $q_0, q_1, ...q_j, ...$ is a run of $\mathcal{A}$ on the sequence of sets of local propositions satisfied by by process i_0 in the path p. In addition this run is an accepting run. Hence, this path satisfies f_{i_0}. As a consequence, we see that $\overline{s}$ satisfies $\vee_i \mathsf{E} f_i$. $\square$

Checking if there exists an infinite path starting from a particular node s and containing infinitely many recurrent nodes is accomplished by checking if there exists a finite path from s to a strongly connected component containing a recurrent node. The later property can be checked using standard graph algorithms that are of linear time complexity in the size of the graph. The number nodes in the graph $\overline{\mathcal{B}}$ is $O(|\overline{\mathcal{S}}|mn)$ where m is the number of states of the automaton $\mathcal{A}$ and n is the number of processes. We can obtain $\mathcal{A}$ using standard tableau construction for PLTL, and in this case m is going to be of order $O(2^{|f_i|})$.

5.3 Model Checking Indexed PLTL

In the previous construction, we used Buchi automata to check properties involving local propositions of a single process together with global propositions. Now, we would like to use Buchi automata for checking properties that may involve local propositions of more than one process together with global propositions. The input alphabet to such automata are subsets of the set of all atomic propositions $\mathcal{AP}$. We define a particular type of automata called *symmetric automata*. First, we need the following definitions. For any $\varphi \subseteq \mathcal{AP}$, and permutation π, we define $\pi(\varphi)$ to be the set $(\varphi \cap \mathcal{AP}') \cup \{P_{\pi(i)} : P_i \in \varphi\}$. Essentially, the $\pi(\varphi)$ is obtained by changing the indices of the local propositions in φ according to the permutation π.

We say that an automaton $\mathcal{A} = (Q, 2^{\mathcal{AP}}, \delta, I, F)$ is symmetric with respect to a group of permutations G if there exists a group action of G on Q, $a : (Q \times G) \to Q$, satisfying the following properties:
For every $q \in Q$ and $\pi \in G$,

- For every $q' \in Q$, and $\varphi \subseteq \mathcal{AP}$, $q' \in \delta(q, \varphi)$ iff $a(q', \pi) \in \delta(a(q, \pi), \pi(\varphi))$.
- $q \in I$ iff $a(q, \pi) \in I$; also, $q \in F$ iff $a(q, \pi) \in F$.

Below, we present a procedure for checking if there exists a path in the original structure $\mathcal{M}$ that is accepted by a symmetric automaton $\mathcal{A}$. We later use this procedure for model checking for a powerful linear temporal logic called Indexed PLTL (in short IPLTL). Let $\mathcal{M} = (\mathcal{S}, \mathcal{R})$ be a structure and $\mathcal{A}$ be an automaton that is symmetric with respect to a group of permutations G, and let a be the function as defined above. First, we construct the annotated quotient structure $\overline{\mathcal{M}} = (\overline{\mathcal{S}}, \mathcal{AR})$ with respect to the group of permutations $Aut\ \mathcal{M} \cap G$. We define a graph $\overline{\mathcal{B}}$ as follows. The nodes of $\overline{\mathcal{B}}$ are pairs of the form $(\overline{s}, q)$ where $\overline{s} \in \overline{\mathcal{S}}$ and $q \in Q$. The set of edges of $\overline{\mathcal{B}}$ is defined below. For every $\overline{s} \in \overline{\mathcal{S}}$, $q \in Q$, transition $(\overline{s}, \pi, \overline{t})$ in $\mathcal{AR}$ and $r \in \delta(q, \varphi)$ where φ is the subset of atomic propositions satisfied in the state $\overline{s}$ in the structure $\mathcal{M}$, there is an edge in $\overline{\mathcal{B}}$ from the node $(\overline{s}, q)$ to the node $(\overline{t}, a(r, \pi^{-1}))$. A *recurrent* node of $\overline{\mathcal{B}}$ is a node of the form $(\overline{s}, q_r)$ where q_r is a recurrent state of $\mathcal{A}$. The following lemma is easily proved from the symmetry property of the automaton.

Lemma 5.4: There exists a path in $\mathcal{M}$ starting from a representative state $\overline{s}$ that is accepted by $\mathcal{A}$ iff there exists an infinite path in $\overline{\mathcal{B}}$ starting from the node $(\overline{s}, q_0)$ where q_0 is an initial state of $\mathcal{A}$, and containing infinitely many recurrent nodes.

We show below how the above lemma can be used for model checking for a powerful indexed temporal logic called IPLTL. In order to define the syntax of IPLTL, we assume that we have two sets of propositions $\mathcal{AP}'$ and $\mathcal{AP}''$ denoting global and local propositions repsectively. All the local propositions are indexed. Let $\mathcal{AP}$ denote $\mathcal{AP}' \cup \mathcal{AP}''$. First we define the set of PLTL formulas over the set of atomic propositions $\mathcal{AP}$. The set of PLTL formulas is the subset of CTL* consisting of all CTL* formulas over $\mathcal{AP}$ that do not use the path quantifier E, i.e. PLTL is the standard linear propositional temporal logic.

IPLTL is the extension of PLTL that allows process quantifiers of the form $\wedge_i$ and $\wedge_{i \neq j}$. The symbols of IPLTL include all those from PLTL together with some index variables such as i, j etc. and the above two types of process quantifiers. We say that an index variable i is free in a formula f if i occurs as the index of a local proposition and this occurrence is not in the scope of a process quantifier of the form $\wedge_i$, or of the form $\wedge_{i \neq j}$ for some j. The set of IPLTL formulas is the smallest set satisfying the following conditions. Every global proposition is an IPLTL formula; if P is a local proposition symbol and i is an index variable then P_i is an IPLTL formula; if f, g are IPLTL formula then $f \wedge g$, $\neg f$, $\mathsf{X} f$, $f\ \mathsf{U}\ g$, and $\wedge_i f$ are IPLTL formulas; if f is an IPLTL formula with free index variable j then $\wedge_{i \neq j} f$ is also an IPLTL formula. A closed IPLTL formula is one that has no free index variables. We fix the set of process indices to be $I = \{1, 2, ..., n\}$. The semantics of IPLTL formulas is defined by translation into PLTL. The translation maps each IPLTL formula f into a PLTL formula f' inductively. The translation is achieved by expanding each process quantifier in the obvious way. It is to be noted that in the resulting formula f', the indices of all local propositions are constants. It can also be shown that for any closed IPLTL formula f, $Aut\ f'$ is going to be the full symmetry set $Sym\ I$. [4]

Given an IPLTL formula f and the annotated structure $\overline{\mathcal{M}}$, we use the following method for checking if all paths in the original structure $\mathcal{M}$ starting from a state s satisfy the formulas. We first construct the automaton $\mathcal{A}$ corresponding to the PLTL formula $\neg f'$. Such an automaton is obtained directly from the tableau associated with $\neg f'$ (see [ES85]). This automaton can be shown to be symmetric with respect to the full symmetry group $Sym\ I$. We construct the product graph $\overline{\mathcal{B}}$ obtained by taking the product of $\overline{\mathcal{M}}$ and the automaton $\mathcal{A}$, and check that there is no infinite path starting from a node of the form $(\overline{s}, q_0)$ that contains infinitely many recurrent nodes where q_0 is the initial state of the automaton $\mathcal{A}$. Clearly, after the annotated quotient structure is constructed, the complexity of the remainder of the procedure is simply proportional to the product of the size of the quotient structure and the size of the automaton $\mathcal{A}$. The size of $\mathcal{A}$ is exponential in the length of f'. The length of f' can itself be exponential in the length of f. However, the complexity of the procedure is going to be proportional to the size of the annotated structure which can be much smaller than the size of the original structure.

6 Example

We now consider a simple example. A solution $\mathcal{P}$ to the mutual exclusion problem is given in Figure 2. Each process K_i has a noncritical section, corresponding to location N_i, and a critical section, represented by location C_i. The transition from N_i to C_i is guarded by the predicate $\wedge_{j \neq i} \neg C_j$. Hence, each process cycles through its two sections preserving the property of mutual exclusion: that no two processes are ever in their critical section at the same time. This can be expressed in CTL by (a formula of the form) $\mathsf{AG}(\wedge_{i \neq j} \neg(C_i \wedge C_j))$. Thus the solution is safe. The starting condition can be captured as $\wedge_i N_i$.

To verify mutual exclusion, for a system with n processes, for any fixed n, we could build its global state transition graph $\mathcal{M}$, with $n+1$ states, as in Figure 3. However, since the communication relation for $\mathcal{P}$ is the complete graph on n nodes, $Aut\ \mathcal{M} = Sym\ [1:n]$. Our rules also tell us that $Aut\ f = Sym\ [1:n]$ Thus we can take $G = Sym\ [1:n]$. Using $(N_1, N_2, \ldots, N_n)$ and $(C_1, N_2, \ldots, N_n)$ as representatives we obtain a quotient $\mathcal{M}/G$ shown in Figure 4. We can now model check over the quotient using theorem 3.3.

7 Related Work

There has been much work done on various bisimulation equivalences and their relationship to program logics. However, none of this work considers automorphisms of a formula as we do and theorem 3.3 was not established in any of the existing works. More over, our paper contains many other results including the formula decomposition, state symmetry and the alternate automata theoretic approach.

The telling quote from Hermann Weyl [We52] in the introduction shows that the basic idea of exploiting the group of automorphisms of a structure in order to understand its basic properties, symmetry and otherwise, is a rather old one in mathematics. However, its application to temporal logic model checking seems to be quite new. In the realm of program verification symmetry seems to have first been utilized, with varying degrees of formality, in the realm of reachability analysis for Petri nets (cf. [JR91]). Here, however, the work seems to have centered around simple reachability ($\mathsf{AG}p$) rather than the full range of temporal correctness properties. Ip and Dill also [ID93] consider the problem of verifying reachability only, but not an arbitrary correctness specification given

[4] Note that $Auto\ f'$ may not be $Sym\ I$.

by a temporal logic formula. Their system provides a new, somewhat more abstract than usual programming language, to facilitate identifications of symmetrys. It has been implemented as the MurΦ system and applied to examples. In [AGS83] and [Ku86] an algebraic approach to reducing the cost of protocol analysis based on the use of quotient structures induced by automorphisms is proposed. For example, the symmetry between 0 and 1 in the alternating bit protocol is factored out to reduce the size of the state space by one half.

The most directly related work is that of Clarke, Filkorn, and Jha [CFJ93] who have independently reported correspondence results similar to those of our section 3 and follow a somewhat similar overall strategy (cf. [BCG89],[St93]). Moreover, they have implemented their ideas using BDD's, provided an analysis of the complexity of BDD-based manipulations of permutation groups showing that testing $\equiv_G$ is graph isomorphism hard for BDD-based representations, and done practical examples. However they do not use formula decomposition, state symmetry, or the alternative automata theoretic approach.

There has been some work done on using symmetries in Petri nets [St91] for computing reachability graphs of nets. However, this work does not consider checking temporal properties over the reduced graph. The work presented in [DBDS93] elegantly combines the symmetry based method with other techniques (such as stubborn sets etc.) to achieve state space reduction in Petri-net based analysis of deadlocks in ADA tasks.

Our work may be distinguished by the most general explicit correspondence results, including CTL* and the Mu-Calculus, and by focusing on the symmetry induced by having many identical processes, which allows us to reduce the difficult problem of computing $Aut\ \mathcal{M}$ to $Aut\ CR$. We also permit auxiliary variables, exploit formula decomposition and state symmetry, and provide an alternative automata-theoretic approach.

8 Conclusions

We have described a framework for expediting model checking by forming the quotient structure modulo a subgroup of the group of automorphisms of the original structure and the specification. The resulting reduction in size can be dramatic when the degree of symmetry is high. The group of automorphisms of the structure depends on process network topology, which is possibly a crucial factor here. For massively parallel systems with with high connectivity and high symmetry like hypercubes, we should get a very good savings. For rings, we would get much less. We have also shown how to improve the efficiency by decomposing large formulae into smaller subformulae. We have further shown that it is possible to exploit the symmetry of individual states to avoid redundant computation. An alternative approach using automata to track shifting indices was also given.

It should be noted that, while we have focused on systems with many isomorphic processes, this is more in the nature of a restriction on the "systems" terminology. Excepting, for example, Theorem 4.1 showing $Aut\ CR \subseteq Aut\ \mathcal{M}$, the basic mathematical machinery here is applicable to systems containing multiple isomorphic "components". All that is really essential is symmetry in the state space, whatever its "physical, systems" source.

At present, we have a method, that is not fully automated. Obviously, we could mechanize it by using naive algorithms to compute automorphism groups, but this in general would not be efficient. Thus important open problems seem to us to be to identify useful special cases for when $Aut\ b$, for various objects b can provably be calculated efficiently, and the related problem of testing $\equiv_G$ efficiently (cf. [CFJ93]). Of course, these are largely group-theoretic in nature. There is a vast literature in computational group theory which should be helpful (cf. [Ho82]). In the interim, we are compiling a catalog of helpful special cases.

Acknowledgments and Historical Remark: We have been thinking about this problem for some time. Actually, we had the Correspondence Lemma 3.1 in 1988 but encountered other difficulties. In any event, we would also like to thank Paul Attie and Steve Kaufman for valuable suggestions. The paper [CFJ93] and an earlier version of our paper, [ES93], were presented at the International Conference on Computer Aided Verification held in Crete, Greece in June 1993. We thank the Programme Committee Chair, Costas Courcabetis, for permitting us to submit our paper later than the officially announced deadline, later, in fact, than all other submissions including [CFJ93]. We also thank Ed Clarke, C. A. R. Hoare, Somesh Jha, and Bob Kurshan for valuable comments on earlier versions.

9 References

[AE89] Attie, P.C., E.A. Emerson, Synthesis of Concurrent Systems With Many Similar Sequential Processes, *Proc. 16th Annual ACM Symp. on Principles of Programming Languages,* Austin, pp. 191–201, 1989.

[APS83] Aggarwal S., Kurshan R. P., Sabnani K. K., "A Calculus for Protocol Specification and Validation", in Protocol Specification, Testing and Verification III, H. Ruden, C. West (ed's), pp. 19–34, North-Holland 1983.

[BCG88] Browne, M. C, Clarke, E. M, and Grumberg, O., Characterizing Kripke Structures in Temporal Logic, *Theoretical Computer Science*, vol. 59, pp. 115–131, 1988.

[BCG89] Browne, M. C, Clarke, E. M, and Grumberg, O., Reasoning about Many Identical Processes, *Inform. and Comp.*, vol. 81, no. 1, pp. 13 – 31, April 1989.

[CE81] Clarke, E. M., and Emerson, E. A., Clarke, E.M., and E.A. Emerson, Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic, *Proc. of the Workshop on Logics of Programs*, Yorktown Heights, D. Kozen, editor, LNCS#131, Springer–Verlag, pp. 52–71, May 1981.

[CES83] Clarke, E.M., E.A. Emerson, and A.P. Sistla, Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach, *Proc. 10th Annual ACM Symp. on Principles of Programming Languages,* Austin, pp. 117–126, 1983; *also appeared in ACM Transactions on Programming Languages and Systems*, **vol.** 8, no. 2, pp. 244–263, 1986.

[CFJ93] Clarke, E. M., Filkorn, T., Jha, S. Exploiting Symmetry in Temporal Logic Model Checking, *Proc. of 5th International Conference on Computer Aided Verification*, Elounda, Greece, pp. 450-462, June 1993.

[DBDS93] Duri, S., Buy U., Devarapalli, R., Shatz. S, Using State Space methods for deadlock analysis in ADA tasking, Proceedings of the 1993 International Symposium on Software Testing and Analysis, pp. 51–60, ACM, June 1993.

[EC82] Emerson, E. A., and Clarke, E. M., Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons, *Science of Computer Programming*, vol. 2, pp. 241 – 266, Dec. 1982.

[Em91] Emerson, E. A., Temporal and Modal Logic, in *Handbook of Theoretical Computer Science*, **vol.** B: *Formal Models and Semantics*, J. van Leeuwen, editor, Elsevier Science Publishers, pp. 995–1072, 1990.

[ES84] Emerson, E. A., and Sistla, A. P., Deciding Full Branching Time Logic, *Information and Control*, Vol. 61, pp. 175-201, June 1984.

[ES93] Emerson, E. A. and Sistla, A. P., Symmetry and Model Checking, *Proc. of 5th International Conference on Computer Aided Verification*, Elounda, Greece, pp. 463–478, June 1993.

[ES94] Emerson, E. A., and Sistla, A. P., "Utilizing Symmetry when Model Checking Under Fairness Assumptions", University of Texas at Austin, Computer Sciences Tech. Report TR-94-17, April 1994.

[He64] Herstein, I, *Topics in Algebra*, Xerox 1964

[Ho82] Hoffmann, C., Graph Isomorphism and Permutation Groups, Springer LNCS no. 132, 1982.

[ID93] Ip, C-W. N., Dill, D. L., Better Verification through Symmetry, Proc. 11th Internartional Symposium on Computer Hardware Description Languages(CHDL), April 1993.

[JR91] Jensen, K., and Rozenberg, G. (eds.), High-level Petri Nets: Theory and Application, Springer-Verlag, 1991.

[Ko78] Kohavi, Z., Switching and Finite Automata Theory, McGraw-Hill, 1978.

[Ku86] Kurshan, R. P., "Testing Containment of omega-regular Languages", Bell Labs Tech. Report 1121-861010-33 (1986); conference version in R. P. Kurshan, "Reducibility in Analysis of Coordination", LNCIS 103 (1987) Springer-Verlag 19-39.

[LY92] Lee, D., Yannakakis M., Online Minimization of Transition Systems, 24th ACM Symposium on Theory of Computing, Victoria, Canada pp. 264–274, 1992.

[MP92] Manna, Z. and Pnueli, A., Temporal Logic of Reactive and Concurrent Systems: Specification, Springer-Verlag, 1992.

[St91] Starke, P. H., Reachability Analysis of Petri Nets Using Symmetries, Syst. Anal. Model. Simul. 8(1991) 4/5, 293-303, Akademic Verlag.

[St93] Stirling, C., Modal and Temporal Logics. in Handbook of Logic in Computer Science, (D. Gabbay, ed.), pp. 1 – 85, Oxford, 1993

[We52] Weyl, H., Symmetry, Princeton Univ. Press, 1952

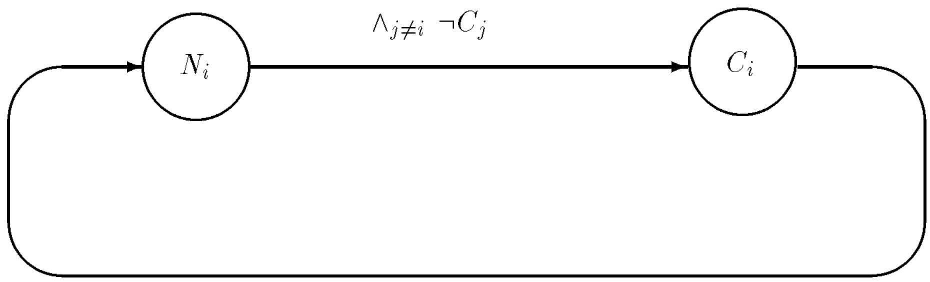

Figure 2: Skeleton for Two State n Process Mutual Exclusion

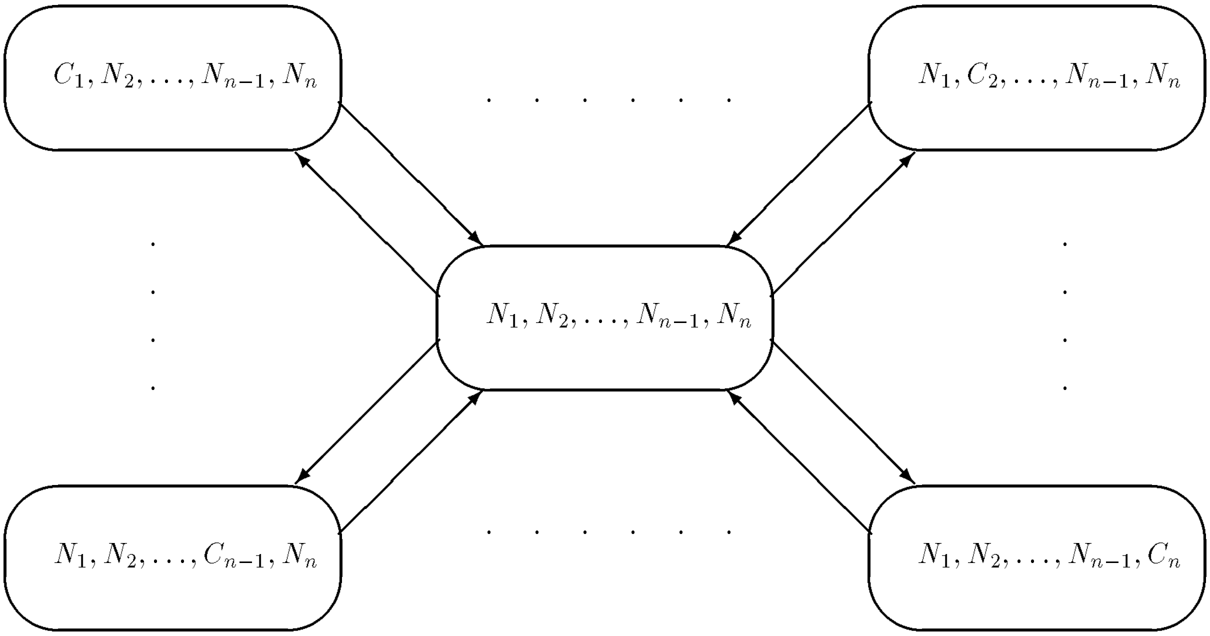

Figure 3: Model for Two State n Process Mutual Exclusion

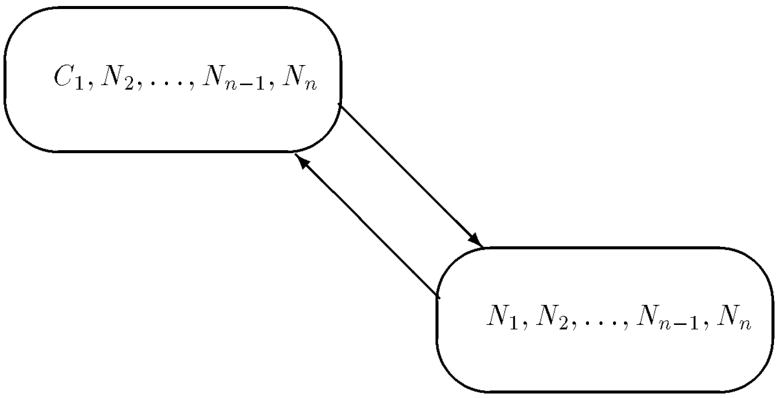

Figure 4: Quotient of Model for Two State n Process Mutual Exclusion