Utilizing Symmetry when Model Checking under Fairness Assumptions

E. A. Emerson[*] A. P. Sistla[†]

May 11, 1994

Abstract: One technique for combating the state explosion problem is to exploit symmetry [12, 5, 9] when performing temporal logic model checking [3, 4]. The works of Clarke, Filkorn, & Jha [5] and Emerson & Sistla [9] show how, using some basic notions of group theory, symmetry may be exploited for the full range of correctness properties expressible in the very expressive temporal logic CTL*. Surprisingly, while fairness properties are readily expressible in CTL*, the methods of [5] and [9] are not powerful enough to admit any amelioration of state explosion, when standard fairness assumptions are involved. In [5, 9] model checking over a large structure M is reduced to model checking over a small quotient structure $\overline{M}$ derived from M by identifying "$\mathcal{G}$-symmetric" states, where $\mathcal{G}$ is a subgroup of permutations on process indices respecting the symmetry of M and leaving invariant the "maximal" propositonal subformulas of the specification f. The latter requirement is crucial but is also the source of the problem. In this paper we will explain why. We will then show how to handle fairness efficiently in spite of this problem.

Let Φ be a fairness constraint. Our alternative, automata-theoretic method depends on showing how to detect the existence of fair paths in the large global state graph M, i.e., testing $M, s_0 \models E\Phi$, using an *annotated* quotient structure $\overline{M}$ defined with respect to a group $\mathcal{G}$ that respects the symmetry of M but does not depend on Φ. In the annotated $\overline{M}$ arcs are labelled by permutations indicating how the meaning of coordinates shift as $\overline{M}$ is uncompressed to obtain M. These permutations are what make it possible for $\overline{M}$ to succinctly encode M and yet provide enough information to model check over $\overline{M}$ (even though $\overline{M}$ does not appropriately respect the symmetry of the specification f or Φ). But they also scramble the meaning of the propositional labelling of states in $\overline{M}$ making it difficult to check for the existence of fair paths. Nonetheless, we show how to efficiently search $\overline{M}$ for any possible "fair" strongly connected subgraphs $\overline{C}$ of $\overline{M}$ that can be unwound into strongly connected subgraphs C of M that contain a path satisfying Φ. To facilitate this search process, we resolve $\overline{M}$ into a *threaded* structure M^* which in essense physically reflects the coordinate shifts in $\overline{M}$ caused by the permutations on arcs. When we find the appropriate "fair" strongly connected subgraphs of M^* then we can conclude that $M, s_0 \models E\Phi$. We can now check whether $M, s_0 \models E(\Phi \wedge f)$ where f is a linear time formula using the automata-theoretic approach of [9]. This in turn makes it possible to efficiently model check $M, s \models g$ where g is a Fair Indexed CTL* formula.

[*]Department of Computer Sciences, The University of Texas at Austin, USA (`emerson@cs.utexas.edu`); Supported in part by ONR contract N00014-89-J-1472, and Texas Advanced Technology Program Grant 003658-250.

[†]Electrical Engineering and Computer Science Department, The University of Illinois at Chicago, USA (`sistla@surya.eecs.uic.edu`); Supported in part by NSF grant CCR-9212183.

1 Introduction

Recently there has been much interest in using various techniques to combat the state explosion problem in the automatic verification of finite state concurrent systems. One of the techniques that has been proposed [12, 5, 9] is to exploit the symmetry inherent in systems with many similar subcomponents when performing temporal logic model checking [3, 4]. In [12] the focus is on reasoning about a simple but basic type of correctness, viz., safety properties expressible in the temporal logic CTL by an assertion of the form $AG\neg error$. The works of Clarke, Filkorn, & Jha [5] and Emerson & Sistla [9] show how, using some basic notions of group theory, symmetry may be exploited for the full range of correctness properties expressible in the very expressive temporal logic CTL*. Surprisingly, while fairness properties are readily expressible in CTL*, the methods of [5] and [9] are not powerful enough to admit any amelioration of state explosion, when standard fairness assumptions are involved.

In this paper, we will explain why fairness is unexpectedly problematic. We will then suggest two potential solutions. The first is a partial solution in which we use a new type of fairness, called *group fairness*, directly in conjunction with the group-theoretic techniques of [5, 9]. Although group fairness is appropriate for some situations, it is not in general adequate. The second solution allows us to handle the full range of conventional fairness properties Φ (cf. [8]). These include *strong fairness* where $\Phi = \wedge_{i=1\ldots n}(GFen_i \Rightarrow GFex_i)$ is the formula[1] asserting that a path is infinite and strongly fair; here the propositions en_i indicate that process i is enabled, while ex_i indicate that process i was just executed. Thus, each process that is enabled infinitely often is executed infinitely often. Similarly $\Phi = \wedge_{i=1\ldots n}(FGen_i \Rightarrow GFex_i)$ captures *weak fairness*, to the effect that any process continuously enabled is executed infinitely often. Note that weak fairness is equivalent, by propositional reasoning, to $\wedge_{i=1\ldots n}GF(ex_i \vee \neg en_i)$. Unconditional fairness is expressed by $\Phi = \wedge_{i=1\ldots n}GFex_i$. Interestingly, this solution depends on trading group theory for automata theory [21].

To understand why fairness in problematic, we must review the "group-theoretic" approaches of [5, 9].[2] We will use the terminology of [9] but the same remarks apply to the the approach of [5]. We focus on systems composed of many homogeneous subcomponents or processes. The global state graph M of such a system may therefore exhibit a great deal of symmetry. The main idea is to reduce, using some basic notions of group theory, model checking over the original, large structure M to model checking over a smaller quotient structure $\overline{M}$ where symmetric states have been identified. In many cases this can yield significant, even exponential savings in the complexity of model checking[3].

The symmetry of M is characterized by the group, *Aut* M, of permutations of process (or subcomponent) indices that define graph automorphisms of M. Let $\mathcal{G}$ be a subgroup of *Aut* M. We define $\overline{M} = M/\mathcal{G}$ to be the quotient structure obtained by identifying any two states s, t of M that are in the same orbit (or equivalence class) of the state space of M, i.e. there exists a permutation π in $\mathcal{G}$ such that $\pi(s) = t$. For example, $s = (N_1, T_2, C_3, T_4)$ might be equivalent to $t = (C_1, N_2, T_3, T_4)$ in a solution to the critical section problem with C_i indicating process i is in its critical section, etc. This permits us to establish the following desired result of [5, 9]:

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

where f is any formula of CTL*, and $\overline{s}$ represents the equivalence class of s – provided also that $\mathcal{G}$ appropriately respects the symmetry of the specification f as well as that of M. The technical

[1]We will assume for ease of exposition that structures are total. So GF means "infinitely often" while FG means "almost always".

[2]In [9] we gave a "group-theoretic" and an "automata-theoretic" approach. Neither is adequate to efficiently handle fairness. Our new approach here generalizes the automata-theoretic approach of [9].

[3]The task of constructing $\overline{M}$ in an efficient manner is addressed in [12, 5, 9] (cf. Section 5).

stipulation is that $\mathcal{G}$ must be a subgroup of $Aut\ M\ \cap\ Auto\ f$ where $Auto\ f$ is the set of permutations that leave invariant all the maximal propositional subformulae of f.

Now suppose $\Phi = \wedge_{i=1\ldots n} GFex_i$ is the unconditional fairness condition. Then $f = E\Phi$ asserts the existence of a fair path. $Auto\ f$ consists of the permutations that leave invariant each of $ex_1, ex_2, \ldots, ex_n$. For each i, the set of permutations leaving ex_i invariant, call it $Stab\ i$, are those that fix or stabilize i and allow the other indices to permute freely. However, the set of permutations that leaves all of the ex_i invariant is $Stab\ 1\ \cap \ldots \cap\ Stab\ n = \{Id\}$ with just the identity permutation. Thus, to model check $f = E\Phi$ the only possible group $\mathcal{G} \subseteq Aut\ M\ \cap\ Auto\ f$ is $\mathcal{G} = \{Id\}$, and this results in no compression whatsoever as $\overline{M} = M$ in this case. In general, when we try to check a CTL* formula involving such a fairness assumption (strong, weak, unconditional fairness are all problematic) we run into this difficulty. The problem is the need to respect the symmetry of the specification in an appropriate way.[4]

To overcome this difficulty, we take an automata-theoretic approach [21] where the original, large Kripke structure M is succinctly represented by an *annotated* quotient structure $\overline{M}$ [9] defined with respect to a group of permutation $\mathcal{G} \subseteq Aut\ M$ that does not depend on the fairness assumption Φ or specification f. $\overline{M}$ has one representative state for each $\mathcal{G}$-equivalence class of M, and its edges are labeled with permutations denoting how coordinates need to be permuted as one traverses that edge. We present efficient algorithms that work on the annotated quotient structure $\overline{M}$ and check for correctness with respect to M of specifications given by Indexed CTL* formulas, under (unconditional, weak or strong) fairness assumptions. It turns out to be necessary and sufficient to solve the fair state problem, i.e. whether $M, s_0 \models E\Phi$, using the annotated quotient $\overline{M}$.

If the algorithm were to operate directly on M, it would look for fair strongly connected subgraphs (i.e., those containing a fair path) reachable from s_0. These can be found efficiently by first computing the maximal strongly connected components C of M. If some C is "obviously fair" – meaning that, for each process index i, if en_i appears in C then ex_i also appears in C – it must be that C is fair, i.e., it contains a fair path. In this case, it is enough to check if there is a path from s_0 to such a C. However, if C is not obviously fair but contains a fair path p, then set of nodes that p visits infinitely often defines an obviously fair strongly connected subgraph D of C. To determine if such a D exists, the algorithm prunes (all such not obviously fair) C by deleting all "bad" nodes in C that are labelled with any en_i for which a node labeled with ex_i does not appear anywhere in C. We let M' be the resulting structure after this pruning and repeat the above process with M' instead of M. After at most n iterations, we will have eliminated all bad en_i nodes and discovered an obviously fair D if it exists.

Of course, we can not work directly with M. Fortunately, we can process $\overline{M}$ in an analogous manner. An essential point is that each strongly connected subgraph C of M collapses to a strongly connected subgraph $\overline{C}$ of $\overline{M}$; conversely, each strongly connected subgraph $\overline{C}$ of $\overline{M}$ can be unwound into a (family of) corresponding strongly connected subgraph(s) C of M. The problem is complicated by the fact that, as $\overline{C}$ in $\overline{M}$ is unwound to obtain C in M, the meaning of propositions is shifted depending on the history of the state on account of the permutation labels on the arcs. For example, assume $\overline{s} - \pi \rightarrow \overline{t}$ is an edge in $\overline{C}$, π is the permutation that swaps 1 with 2, $\overline{s}$ is labelled with en_1 and ex_2, and $\overline{t}$ is labelled with en_2 and ex_1. Then we can unwind C starting at $\overline{s}$ along that edge to get the edge $\overline{s} \longrightarrow \pi(\overline{t})$ in C. Because of the transposition π, en_2 at $\overline{t}$ in $\overline{C}$ then represents en_1 at state $\pi(\overline{t})$ of C.

These coordinate shifts are what make it possible to for $\overline{M}$ to succinctly encode M and yet pro-

[4]It is *not* adequate that $\mathcal{G}$ be a subgroup of just $Aut\ f$, the permutations respecting top level formula f.

vide enough information to model check over $\overline{M}$, even though $\overline{M}$ does not appropriately respect the symmmetry of of the specification or Φ. But they also scramble the "meaning" of propositions as suggested above. To efficiently keep track of such coordinate shifts, we unravel $\overline{M}$ into its constituent *threads*. That is, we form a "threaded" structure M^* whose nodes are essentially the individual coordinates of the nodes in $\overline{M}$ and whose edges "physically" reflect the permutations on coordinates. For instance, the edge $\overline{s} - \pi \longrightarrow \overline{t}$ above is broken down into two thread edges $(\overline{s}, en_1) \longrightarrow (\overline{t}, en_2)$ and $(\overline{s}, ex_2) \longrightarrow (\overline{t}, ex_1)$. For each strongly subgraph $\overline{C}$ this induces a subgraph C^* of threads, permitting us to test whether $\overline{C}$ represents a fair C by checking the fair components of C^*.

It is worth noting that the basic automata-theoretic approach over annotated structures (cf. [9]), which itself makes no special provisions for fairness, can be directly applied in an effort to handle fairness efficiently. The idea is that, to check $M, s_0 \models Ef$, we form the product[5] graph (automaton) $\overline{B}$ of $\overline{M}$ with $\mathcal{B}_f$, where $\mathcal{B}_f$ is an automaton on strings obtained by modifying $\mathcal{A}_f$, the automaton for linear time formula f derived by the usual tableau construction, to account for the shifting meaning of coordinates using the permutations on arcs of $\overline{M}$ as guides. We can then give an algorithm to answer whether $M, s_0 \models Ef$ by testing $\overline{B}$ for nonemptiness in time polynomial in $|\overline{M}|$ and $|\mathcal{B}_f|$ (cf. [16, 21]). For a formula f with a single index i, $\mathcal{B}_f$ is essentially the same automaton as $\mathcal{A}_f$ but its states are a pair of the form (q, j) where q is a state of $\mathcal{A}_f$ and j is an index giving the current meaning ("location") of index i. If we think of $\mathcal{B}_f$ reading a path through $\overline{M}$ it applies the transition function of $\mathcal{A}_f$ to update q using coordinate j for the source input symbol and then updates j based on the next permutation π it traverses. However, if f involves k indices, then $\mathcal{B}_f$ must store a tuple $(j_1 \ldots j_k)$ of permuted indices. Thus, $|\mathcal{B}_f| = O(|\mathcal{A}_f| \cdot n^k)$. This is of polynomial size for fixed, bounded k. However, when $f = \Phi$ there are n indices involved, which causes its state space to be of size exponential in n. Since M itself is typically of size exponential in n, this defeats our purpose.

Using our new efficient method to handle the fairness constraint Φ over $\overline{M}$ together with our automata theoretic approach above to handle a linear time specification f over $\overline{M}$, we show how to check $M, s_0 \models E(\Phi \wedge f)$ in time linear in the size $|\overline{M}|$ of the annotated quotient and polynomial in $|\Phi|$ and $|\mathcal{B}_f|$. By recursive descent [8], this yields efficient model checking algorithms for Fair Indexed CTL*, which permits indexed CTL* assertions with "process modalities" ranging over (a bounded number of) indices and path quantifiers (A_Φ, E_Φ) ranging over fair paths.[6] We can say $\wedge_i A_\Phi(G(T_i \Rightarrow FC_i))$ – along all fair paths each process is free from individual starvation. Also allowed are such formulas as $\wedge_{i \neq j} E_\Phi G(C_i \wedge C_j)$ for potential "collision" among any pair of processes.

This paper is organized as follows. Section 2 contains the definitions and notation used in the paper. Section 3 discusses the problem with the previous approaches and shows how the previous methods can be used for checking correctness under group fairness. Section 4 describes the algorithms for standard (strong, weak, unconditional) fairnesses using annotated quotient structures. Section 5 contains concluding remarks.

2 Preliminaries

We assume that we have a system of n processes communicating through shared variables. We let $I = \{1, 2, ..., n\}$ be the set of process indices. We let V denote the set of variables in the system. We represent a variable X that is shared among a set I' of processes by $X_{I'}$. For example if X is shared

[5]We find it convenient inside to define $\overline{B}$ as the product of $\overline{M}$ and $\mathcal{A}_f$ using a slightly different "multiplication" yielding the same result. In fact, we can construct a threaded B^* directly without building $\overline{B}$.

[6]Our formulation of Indexed CTL* is slight variant of that in [7].

between processes 1 and 2, it is represented by $X_{\{1,2\}}$ (actually, we write this as $X_{1,2}$); if another variable with the same name X is shared between processes 3 and 4, then it is represented by $X_{3,4}$. All variables of the form Y_i are local variables of processes i. We let V denote the set of variables in the system. A global state s is a function that assigns values to all the variables in V.

The operational behavior of such a system can be modeled by a Kripke structure $M = (S, R, L, S_0)$ where S is the set of global states, R is a (total) binary relation giving the one step transitions of the system, L is a labelling function that denotes the atomic propositions that are satisfied in each global state, and S_0 is the set of initial states.

We are interested in defining symmetries on such a system of processes. We let *Sym* I be the set of all permutations π on the set I of indices. *Sym* I forms a group with functional composition being the group operation. We let Id denote the identity permutation and π^{-1} the inverse of π. For any variable $X_{I'}$ in V, we let $\pi(X_{I'})$ denote the variable $X_{\pi(I')}$ where $\pi(I')$ is obtained by replacing each element i in I' by $\pi(i)$. Note that $\pi(X_{I'})$ need not be a variable in V. However, we say that π *respects* V if for every variable $Y \in V$, $\pi(Y)$ is also in V. It is not difficult to see that π respects V iff π is a one-one and on-to function , i.e. is an automorphism on V.

We say that a permutation π on I is an automorphism on M if the following conditions are satisfied:

- π respects V.
- For every pair of states s and t in S, $(s,t) \in R$ iff $(\pi(s), \pi(t)) \in R$.
- For every s, $s \in S_0$ iff $\pi(s) \in S_0$.

We denote the set of automorphisms of M by *Aut* M. It is not difficult to see that *Aut* M is a subgroup of *Sym* I. Let $\mathcal{G}$ be any subgroup of *Aut* M. We say that two states s and t in S are equivalent with respect to $\mathcal{G}$, written $s \equiv_{\mathcal{G}} t$ if there exists a $\pi \in \mathcal{G}$ such that $t = \pi(s)$. Since $\mathcal{G}$ is a group, $\equiv_{\mathcal{G}}$ is an equivalence relation.

Our specification logics use a set of atomic propositions. We assume that we have two types of atomic proposition symbols— local symbols and global symbols. If P is a local proposition symbol then P_i, for each $i = 1, ...n$, is going to be a local atomic proposition. A global symbol Q is simply a global atomic proposition. Thus, all local atomic propositions are indexed with process indices while global atomic propositions are not.

The labelling function L in M assigns atomic propositions for each state in S. We require that L should satisfy the following conditions:

- For every global atomic proposition Q, for every s and for every $\pi \in$ *Aut* M, s satisfies Q (i.e. $Q \in L(s)$) iff $\pi(s)$ satisfies Q.
- For every local atomic proposition P_i, for every s and for every $\pi \in$ *Aut* M, s satisfies P_i iff $\pi(s)$ satisfies $P_{\pi(i)}$.

We are interested in checking properties specified in various logics. PLTL is the standard propositional linear temporal logic built up from atomic propositions, boolean connectives, and the usual linear time operators G (always), F (sometime), X (next time), and U (until). CTL* is the logic that extends PLTL by also allowing the path quantifiers A (for all fullpaths) and E (for some fullpath) to also be used. The *basic modalities* of CTL* are formulae of the form Ef where f is a pure PLTL formula. All CTL* formulae can be obtained by taking boolean combinations and nestings of the basic modalities. CTL is a restricted version of CTL*. Indexed CTL* is built up from basic modalities of the form $\vee_i Ef_i$, $\wedge_i Ef_i$, $\vee_{i\neq j} Eg_{i,j}$, and $\wedge_{i\neq j} Eg_{i,j}$, where f_i and respectively $g_{i,j}$ are PLTL formulas that use only global atomic propositions and/or local atomic propositions of index i or respectively i and j; $\vee_i$, $\wedge_i$ act as existential and universal process quantifiers ranging over single process indices, while $\vee_{i\neq j}$, $\wedge_{i\neq j}$ range over pairs of distinct indices. Formulas of Indexed CTL* are inductively built up from the basic modalities using boolean connectives and nesting (an Indexed CTL* formula may be substituted for a global proposition in another Indexed CTL* formula). Fair Indexed CTL* is just like Indexed CTL* but uses the path quantifiers E_Φ and A_Φ where path quantification ranges only over fair paths [8]. The semantics of these logics is defined in the usual way [10] and we write, e.g., $M, s \models h$ to mean that in structure M at state s formula h holds true.

Let f be any formula f of the above logics. We let $\pi(f)$ denote the formula obtained by changing the indices of local atomic propositions in f according to the permutation π. In [9], we defined to groups of permutations $Aut\ f$ and $Auto\ f$ that capture symmetries in the formula f. $Aut\ f$ is the set of permutations π such that f is equivalent to $\pi(f)$. $Auto\ f$ is a subgroup of $Aut\ f$, and is the set of permutations that leave invariant all maximal (under the subformula relation) propositional subformulas of f.

We assume that we have two special local propositional symbols en and ex. For any i, the local atomic proposition en_i is satisfied in a state s iff process i is enabled in s; the atomic proposition ex_i is satisfied in a state s iff all transitions leading to s, i.e. all transitions of the form (s', s) in R, are due to the execution of a single step of process i. This would require that, for each state s, there is exactly one value of i, such that ex_i is satisfied in s. It should be obvious to see that for any $\pi \in\ Aut\ M$, ex_i is satisfied in s iff $ex_{\pi(i)}$ is satisfied in $\pi(s)$ and similarly for en_i. We say that an infinite path p of M is *strongly fair* if it satisfies $\Phi = \wedge_{i=1\ldots n}(GFen_i \Rightarrow GFex_i)$, meaning that each process that is enabled infinitely often is executed infinitely often. Similarly a path p is *weakly fair* if it satisfies $\Phi = \wedge_{i=1\ldots n}(FGen_i \Rightarrow GFex_i)$ meaning that any process that is continusly enabled. continuously enabled is executed infinitely often. Note that weak fairness is equivalent, by propositional reasoning, to $\wedge_{i=1\ldots n}GF(ex_i \vee \neg en_i)$. A path is *unconditionally fair* if it satisfies $\Phi = \wedge_{i=1\ldots n}GFex_i$, meaning simply that each process is executed infinitely often.

3 Checking correctness under group fairness

In earlier papers [5, 9], it was shown how model checking for CTL* formulas over the structure M can be reduced to modelchecking over a quotient structure $\overline{M}$. We state the result as given in [9]. Let f be a CTL* formula and $\mathcal{G}$ be a subgroup of $Aut\ M\ \cap\ Auto\ f$. The quotient structure $\overline{M}$ of $M = (S, R, L, S_0)$ with respect to $\mathcal{G}$, has a single state representing each equivalence class in S. For a state $s \in S$, we let $\overline{s}$ denote the representative of the equivalence class of s, i.e. $\overline{s} \equiv_\mathcal{G} s$.

Formally, $\overline{M} = (\overline{S}, \overline{R}, \overline{L}, \overline{S}_0)$, where for each equivalence class of $\equiv_\mathcal{G}$, there a single state representing that equivalence class in $\overline{S}$. If $\overline{s}$ and $\overline{t}$ are states in $\overline{S}$, then $(\overline{s}, \overline{t}) \in \overline{R}$ iff for some states s, t in S, such that $s \equiv_\mathcal{G} \overline{s}$ and $t \equiv_\mathcal{G} \overline{t}$, $(s, t) \in R$. The function $\overline{L}$ is a restriction of L to the states in $\overline{S}$, and $\overline{S}_0 = \{\overline{s} : \text{for some } s \in S_0\ \overline{s} \equiv_\mathcal{G} s\}$.

It was shown in [9] that for any state s, the CTL* formula f is satisfied at state s in M iff it is satisfied at the state $\overline{s}$ in $\overline{M}$. However, this result is not useful for checking correctness properties under fairness assumptions, particularly liveness properties. Let f be a "Fair CTL*" formula of the form $A_\Phi g$ where g is a PLTL formula and Φ is, say, unconditional fairness $\wedge_{i=1\ldots n}GFex_i$. f is equivalent to the CTL* formula $f' = A(\Phi \Rightarrow g)$. We are interested in checking if $M, s \models A_\Phi g$, i.e., if along all fair paths in m starting at s does g hold true. In order to reduce model checking of f' in the original structure M to modelchecking over a quotient structure, we need to use a group $\mathcal{G}$ which is a subgroup of $Aut\ M \ \cap \ Auto\ f'$. Unfortunately, from the definition of $Auto$, we calculate that $Auto\ f' = Auto\ \Phi = \{Id\}$ where Id is the identity permutation. Thus, $G = \{Id\}$ and the quotient structure with respect to $\mathcal{G}$ will be identical to the original structure. As a consequence, we do not get any reduction in the state space. The same problem occurs even if we use the approach as given in [5].

However, we show that using the above approach we still can check for correctness under a notion called *group fairness.* As given in the previous paragraph, let $f = A(g)$ where $\mathcal{G}$ is a PLTL formula, and let $\mathcal{G}$ be any subgroup of $Aut\ M \cap Auto\ f$. Let $\overline{M}$ be the quotient structure of M with respect to $\mathcal{G}$. Now, we define an equivalence relation $\sim$ among the process indices I as follows. For any $i, j \in I$, $i \sim j$ iff there exists a permutation $\pi \in \mathcal{G}$ such that $j = \pi(i)$. It can easily be seen that $\sim$ is an equivalence relation. Let $I_1, I_2, ..., I_k$ be all the equivalence classes of $\sim$. For each $i = 1, ...k$, let C_i denote the propositional formula $(\vee_{j \in I_i} ex_j \ \vee \ \wedge_{j \in I_i} \neg en_j)$. Essentially, C_i states that some process in the set I_i is executed or all processes in the set I_i are disabled. We say that a path p in M is *weakly fair with respect to the group* $\mathcal{G}$ iff for each $i = 1, ..., k$, C_i is satisfied infinitely often on p. This fairness condition is equivalent to the following condition: if for each $i = 1, ...k$, if all processes in the set I_i are disbaled infinitely often, or some process in I_i is executed infinitely often.

Now, let Φ_{gp} denote the formula $\bigwedge_{i=1...k} GFC_i$, and $f' = A(\Phi_{gp}; \Rightarrow g)$. Now, it should be obvious to see that all paths in M that start from s and that are weakly fair with respect to the group $\mathcal{G}$ satisfy the PLTL formula $\mathcal{G}$ iff the formula f' is satisfied at s in M. It can also be shown that $Auto\ f' = G$ and hence $\mathcal{G}$ is a subgroup contained in $Aut\ M \ \cap \ Auto\ f'$. Now using the the results of [9], it is easy to see that f' is satisfied at s in M iff f' is satisfied at $\overline{s}$ in $\overline{M}$. This result shows that we can exploit symmetry for checking correctness under fairness as long as the fairness is with respect to a group $\mathcal{G}$. This is a form of fairness which could be useful in some applications. We also note that we can define an analogous notion of strong fairness with respect to group $\mathcal{G}$ that can be handled efficiently in the same way.

4 Checking correctness under standard fairnesses

In this section we give efficient algorithms for model checking formulas of Fair Indexed CTL*. In our earlier paper [9], we gave efficient algorithms for checking Indexed CTL* formulas where the path quantifiers (A, E) range over all paths. The algorithms in the automata-theoretic portion of that work use annotated quotient structures as we will here. As explained in the introduction, because a fairness constraint Φ is expressible in CTL*, the previous automata-theoretic approach could be applied directly to reason under fairness assumptions but it would not be efficient. Here in this section, we show how to efficiently handle fairness over annotated quotient structures. This then permits us to model check Fair Indexed CTL* where the path quantifiers (A_Φ, E_Φ) range over fair paths. We assume that the fairness constraint Φ is strong fairness. This also permits us to handle unconditional and weak fairness since they are special cases of strong fairness [8].

LEMMA 4.1: Two ($\mathcal{G}$-)equivalent states in M satisfy the same set of Fair Indexed CTL* formulas.

Proof: We can argue by induction on formula structure that for any formula f of Fair Indexed CTL*, $Aut\ f = Sym\ I$. Assume $s \equiv_\mathcal{G} t$ so that $t = \pi(s)$ for some $\pi \in \mathcal{G}$. We have $M, s \models f$ iff $M, \pi(s) \models \pi(f)$ (since $\pi \in \mathcal{G} \subseteq \ Aut\ M$) iff $M, t \models f$ (since $t = \pi(s)$ and $Aut\ f = Sym\ I$). □

The crucial step in the model checking problem is to handle the basic modalities [8]. We will describe in detail how to handle $\vee_i E_\Phi f_i$ and $\wedge_i E_\Phi f_i$. Treatment of $\vee_{i\neq j} E_\Phi g_{i,j}$ and $\wedge_{i\neq j} E_\Phi g_{i,j}$ is analogous but the algorithm must keep track of two coordinates rather than one. By Lemma 4.1, it is therefore enough to check satisfaction of such basic modalities at representative states.

As indicated in the introduction, we will use annotated quotient structures for model checking. First, we define the annotated quotiented structure. The annotated quotient structure is similar to the quotient structure except that the edges of the structure are labeled with permutations indicating how the coordinates need to be permuted. As before, let $M = (S, R, L, S_0)$ be the Kripke structure modeling the behavior of the system of processes. Let $\mathcal{G}$ be a subgroup of $Aut\ M$. We keep $\mathcal{G}$ fixed through out this section. The annotated quotient structure $\overline{M}$ with respect to $\mathcal{G}$, is the quadruple $(\overline{S}, AR, \overline{L}, \overline{S}_0)$ where $\overline{S}, \overline{L}$ and $\overline{S}_0$ are as defined in the previous section; AR is a set of triples of the form $(\overline{s}, \pi, \overline{t})$ such that there is an edge from $\overline{s}$ to $\pi(\overline{t})$ in M, i.e. $(\overline{s}, \pi(\overline{t})) \in R$. An annotated path in $\overline{M}$ is a finite or an infinite alternating sequence $\overline{s}_0, \pi_1, \overline{s_1}, \pi_2, ..., \pi_i, \overline{s_i}, ...$ of states and permutations such that for each $i > 0$ $(\overline{s_{i-1}}, \pi_i, \overline{s_i}) \in AR$; if the sequence is finite then it should end in a state. For each annotated path p, we define a sequence $f(p)$ of states in S such that $f(p) = t_0, t_1, ..., t_i, ...$ where $t_0 = \overline{s_0}$ and $t_i = \pi_1 \cdot \pi_2 \cdots \pi_i(\overline{s_i})$. The following lemma, proved in [9], relates annotated paths in $\overline{M}$ to paths in M.

LEMMA 4.2: For every annotated path p in $\overline{M}$, $f(p)$ is a path in M. In the other direction, for every path q starting at a representative state in M, there exists an annotated path p in $\overline{M}$ such that $f(p) = q$.

We will be using automata for model checking temporal properties. A Buchi automaton $\mathcal{A}$ on infinite strings is a quintuple $(Q, \Sigma, \delta, init, F)$ 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, $init \in Q$ is the *initial* state and $F \subseteq Q$ is the set of *final* 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 is the initial state $init$, and for all $i \geq 0$, $q_{i+1} \in \delta(q_i, t_i)$. We say that a run is accepting iff some final 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 strong fair 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 define a directed graph B which is a cross product of M and $\mathcal{A}$ as follows. Our algorithm does not construct the graph B. We only use it in our proofs. The nodes/states of B are triples of the form (s, q, i) where $s \in S$, $q \in Q$ and $i \in I$. Formally, $B = (V, E)$ where $V = S \times Q \times I$ is the set of nodes and E is the set of transitions/edges of B. There is an edge from node (s, q, i) to the node (s', q', i') iff $i = i'$, $(s, s') \in R$ and in addition, there is a transition of the automaton from state q to the state q' on the input which is the set of global propositions satisfied in state s and the set of local propositions that are satisfied by process i's component of s. A node (s, q, i) in B is called a *final* node iff q is a final state of the automaton $\mathcal{A}$. Intuitively, B denotes the simulation of the automaton $\mathcal{A}$ on the execution of different processes indicated by the process index i in each node. It is to be noted that all edges in B are between nodes with the same process index. Let π be any permutation on I and (s, q, i) be any node of B. We also define a labeling function L_B that labels each node of B with local atomic propositions en_i, ex_i for each $i = 1, .., n$. For any node (s, q, i), $L_B((s, q, i))$ contains en_i (resp., ex_i) iff $en_i \in L(s)$ (resp. $ex_i \in L(s)$). We say that an infinite path p in B is strongly fair iff the following condition is satisfied: for each $i = i, ..., n$ if p contains infinitely many nodes that satisfy en_i then it also contains infinitely many nodes that satisfy ex_i.

LEMMA 4.3: The following properties hold for all $s \in S$.

- $\vee_i E_\Phi f_i$ holds at state s of the structure M iff for some i, $1 \leq i \leq n$, there exists a path in B starting at node $(s, init, i)$ satisfying Φ and containing infinitely many final nodes.
- $\wedge_i E_\Phi f_i$ holds at state s of the structure M iff for all i, $1 \leq i \leq n$, there exists a path in B starting at node $(s, init, i)$ satisfying Φ and containing infinitely many final nodes.

A strongly connected component (SCC) C in a directed graph is a total subgraph such that there is a path from every node in C to every other node which only passes through the nodes in C. A maximal strongly connected component (MSCC) is a strongly connected component such that no strict super set of it is an SCC.

Let C be a SCC in B. We say that C is strongly fair iff the following condition is satisfied for each $i = 1, ..., n$: If C contains a node of the form (s, q, j) such that $en_i \in L(s)$, i.e. process i is enabled in s, then C also contains a node of the form (t, q', j) such that $ex_i \in L(t)$. We say that C is a final SCC if it contains a node of the form (s, q, i) where q is a final state of the automaton.

LEMMA 4.4: There exists a path in M starting at state s that satisfies the PLTL formula f_i and the strong fairness constraint Φ iff there exists a path in B from $(s, init, i)$ to a final SCC that is strongly fair.

For any permutation π on I, we define $\pi((s, q, i))$ to be the node $(\pi(s), q, \pi(i))$, i.e. the permutation π applied to the node (s, q, i) changes s and i according to π, while keeping the automaton state unchanged. It is easy to see that for every $\pi \in$ *Aut* M, there is an edge from (s, q, i) to (s', q', i') iff there is an edge from $\pi((s, q, i))$ to $\pi((s', q', i'))$. Hence every $\pi \in$ *Aut* M is also an automorphism of B.

We next construct another structure $\overline{B} = (\overline{V}, \overline{E})$ which is a product of the annotated structure and the automaton $\mathcal{A}$. Here $\overline{V} = \overline{S} \times Q \times I$ is the set of states/nodes. The set of transitions $\overline{E}$ which consists of triples of the form (x, π, y) ,where $x, y \in \overline{V}$, is defined as follows. For every transition of the form $(\overline{s}, \pi, \overline{t}) \in AR$ and for every automaton state q and process index j, there is a transition (x, π, y) in $\overline{E}$ where $x = (\overline{s}, q, j)$, $y = (\overline{t}, r, \pi^{-1}(j))$, and 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 $\overline{s}$. It is to be noted that the transitions in $\overline{B}$ can be between nodes with different process indices, while this is not the case in B. We say that a state $(\overline{s}, q, j)$ of $\overline{B}$ is a *final* state iff q is a final state of $\mathcal{A}$. Recall that *init* is the initial state of $\mathcal{A}$. We define an annotated path in $\overline{B}$ exactly similar to the annotated paths in $\overline{M}$.

An annotated path in $\overline{B}$ is a finite or an infinite alternating sequence $x_0, \pi_1, x_1, \pi_2, ..., \pi_i, x_i, ...$ of states in $\overline{V}$ and permutations such that for each $i > 0$ $(x_{i-1}, \pi_i, x_i) \in \overline{E}$; if the sequence is finite then it should end in a state. For each annotated path p, we define a sequence $f(p)$ of nodes in V such that $f(p) = t_0, t_1, ..., t_i, ...$ where $t_0 = x_0$ and $t_i = \pi_1 \cdot \pi_2 \cdots \pi_i(x_i)$. For a finite annotated path $p = x_0, \pi_1, x_1, ..., \pi_k, x_k$, let $\pi(p)$ denote the permutation $\pi_1 \cdot \pi_2 \cdot ... \cdot \pi_k$.

The following lemma relates the annotated paths in $\overline{B}$ to paths in B.

LEMMA 4.5: For every annotated path p in $\overline{B}$, $f(p)$ is a path in B. In the other direction, for every path q in B, there exists an annotated path p in $\overline{B}$ such that $f(p) = q$.

A strongly connected component C of $\overline{B}$ is a set of nodes such that for every pair of nodes x, y in C there exists a finite annotated path from x to y that passes only through the nodes in C. Recall that for any state $s \in S$, $\overline{s}$ denotes the unique representative of the equivalence class containing s; the equivalence classes that we consider are those induced by the equivalence relation $\equiv_{\mathcal{G}}$. For any node $x = (s, q, i) \in V$, let $\overline{x}$ denote the node $(\overline{s}, q, \pi^{-1}(i))$ where π is the permutation that maps $\overline{s}$ to s, i.e. $\pi(\overline{s}) = s$. For any set of nodes $C \subseteq V$, let $\overline{C} = \{\overline{x} : x \in C\}$. Also, for any strongly connected component D in $\overline{B}$ and any $x \in D$, define $g(x, D)$ to be the set of all nodes of the form $\pi(p)(x)$ where p is a finite annotated path in $\overline{B}$ starting from x and containing only nodes in D. The following lemma shows that there is a correspondence bewteen the SCCs in B and the SCCs in $\overline{B}$.

LEMMA 4.6: For every SCC C in B, $\overline{C}$ is an SCC in $\overline{B}$. Similarly, for every SCC D in $\overline{B}$ and node $x \in D$, $g(x, D)$ is an SCC in B.

Now, we define the "threaded" directed graph $B^* = (V^*, E^*)$ and a labeling function L^* as follows. The nodes in V^* are pairs of the form (x, i) where $x \in \overline{V}$ and $i \in I$. There is an edge from (x, i) to (y, j) in E^* iff there exists a permutation $\pi \in \mathcal{G}$ such that $j = \pi^{-1}(i)$ and $(x, \pi, y) \in \overline{E}$. The labeling function L^* labels each node in V^* with the local propositional symbols ex and en as follows. Let (x, i) be a node in V^* where $x = (\overline{s}, q, j)$ for some $q \in Q$ and $j \in I$. Then, $ex \in L^*((x, i))$ iff $ex_i \in L(\overline{s})$, and $en \in L^*((\overline{s}, i))$ iff $en_i \in L(\overline{s})$. The node (x, i) as given above is called a *final* node iff the automaton state q is a final state.

Now, we state a simple lemma that shows correspondence between path in B^* and annotated paths in $\overline{B}$. We say that an annotated $p = x_0, \pi_1, x_1, ..., \pi_k, x_k$ in $\overline{B}$ *corresponds* to the path $(x_0, i_0), (x_1, i_1), ..., (x_k, i_k)$ in B^* iff for every j such that $0 < j \leq k$, $i_j = (\pi_1 \cdot \pi_2 \cdot ... \cdot \pi_k)^{-1}(i_0)$.

LEMMA 4.7: If p is an annotated path in $\overline{B}$ starting from the node x_0, then for every $i = 1, ..., n$ there is a path in B^* starting from (x_0, i) that corresponds to p. In the other direction, for every path in B^* starting from a node of the form (x_0, i) there exists a corresponding augmented path in B^* starting from x_0.

Let C be a SCC in B^*. C is said to be *strongly fair* if the following condition is satisfied: if C contains a node labeled (i.e. labeled by L^*) with en then it also contains a node labeled with ex. We say that C is a *final* SCC if it contains a final node.[7]

For any $D \subseteq \overline{V}$, we let B^*/D denote the restriction of B^* to all nodes in V^* of the form (x, i) where $x \in D$, i.e. B^*/D is obtained by deleting from B^* all nodes of the form (y, i) where $y \notin D$.

LEMMA 4.8: Let D be an SCC in $\overline{B}$. Then, all the maximal SCCs in B^*/D are disconnected (i.e. B^*/D has no edges that connect nodes in different MSCCs). Furthermore, for each $x \in D$ and for each MSCC F of B^*/D, there exists an i such that $(x, i) \in F$.

Proof: Let D be a SCC in $\overline{B}$. It is enough if we prove the following: For every edge (x', y') in B^*/D both x' and y' belong to the same SCC. Let (x', y') be any edge in B^*/D and let $x' = (x, i)$ and $y' = (y, j)$. By definition, there exists a transition of the form (x, π_1, y) in $\overline{E}$ such that $j = \pi_1^{-1}(i)$. Since x and y belong to the SCC D, it follows that there exists a cycle in $\overline{B}$ starting with the above transition, i.e. there exists an augmenetd path $p = x_0, \pi_1, x_1, \pi_2, ..., \pi_k, x_k$ in $\overline{B}$ such that $x_0 = x_k = x$, $x_1 = y$ and all the nodes on p are in D. Now consider $\pi(p)$, i.e. $\pi(p) = \pi_1 \cdot ... \cdot \pi_k$. Clearly there exists an integer $r > 0$ such that $(\pi(p))^r = Id$ where Id is the identity permutation. Using lemma 4.7, we see that the annotated path p^r gives us a cycle in B^*/D that passes through x' and y'.

To prove the second part of the lemma, we use the following observation: if there is an annotated path from x to y in $\overline{B}$ passing through the nodes in D then, for every $j = 1, ..., n$ there exists an i such that there is a path from (x, i) to (y, j) in B^* passing through nodes of the form (z, k) where $z \in D$. Now consider any MSCC F in B^*/D. Let (y, j) be any node in F. From our previous observation it follows that, for some i, (x, i) has a path to (y, j) in B^*. From the first part of the lemma it follows that $(x, i) \in F$. $\square$

[7]A final SCC is not to be confused with a terminal maximal strongly connected component.

LEMMA 4.9: Let D be any SCC in $\overline{B}$. Then, for any $\overline{x} \in D$ the SCC $g(\overline{x}, D)$ in B is a final SCC and is strongly fair, iff, all the maximal SCCs in B^*/D are also final SCCs and are strongly fair.

Proof: Let D be any SCC in $\overline{B}$. For each $i = 1, ..., n$, we will show that there exists a node in $g(\overline{x}, D)$ that satisfies en_i (respectively, ex_i) iff there exists a node u labeled with en (respectively, ex) in the MSCC of B^*/D that contains $(\overline{x}, i)$. Now fix an i such that $1 \leq i \leq n$. Now consider any node y in B and let $\pi \in \mathcal{G}$ be the permutation such that $\pi(\overline{y}) = y$. ¿From lemma 4.5, we see that y is in the SCC $g(\overline{x}, D)$ iff there exists an annotated path p in D from $\overline{x}$ to $\overline{y}$, such that $\pi(p)(\overline{y}) = y$; Recall that $\pi(p)$ is the product of permutations that appear on the augmenetd path. From lemma 4.7, we see that there exists a path in B^*/D from $(\overline{x}, i)$ to $(\overline{y}, j)$ iff there exists an augmeneted p in D from $\overline{x}$ to $\overline{y}$ where $j = (\pi(p))^{-1}(i)$. Putting both of these together we see that there is a path in $g(\overline{x}, D)$ from $\overline{x}$ to y iff there exists a path in B^*/D from node $(\overline{x}, i)$ to $(\overline{y}, j)$ where $j = \pi^{-1}(i)$. Since $y = \pi(\overline{y})$, it follows that the node y satisfies the atomic proposition en_i (respectively, ex_i) iff the node $\overline{y}$ satisfies en_j (respectively, ex_j). ¿From the way we defined L^*, it follows that the node $(\overline{y}, j)$ is labeled with en (resp., ex) iff y satisfies en_i (resp., ex_i). Using lemma 4.8, we see that there is a path from $(\overline{x}, i)$ to $(\overline{y}, j)$ in B^*/D iff both of them are in the same MSCC in B^*/D. Putting all the above observations together, it is easy to see that, for each $i = 1, ..., n$, $g(\overline{x}, D)$ contains a node that satisfies en_i (respectively, ex_i) iff the MSCC in B^*/D containing $(\overline{x}, i)$ also contains a node labeled with en (resp., ex). In addition, it should be easy to see that, for every $\overline{z} \in D$, every MSCC in B^*/D contains a node of the form $(\overline{z}, k)$ for some k. Hence $g(\overline{x}, D)$ is a final SCC iff every MSCC in B^*/D is also a final SCC. Now, the lemma follows from the previous arguments. $\square$

Now, we present the algorithm that determines all representative states that satisfy the Fair Indexed CTL* formula $\vee_i E_\Phi f_i$ under strong fair semantics. We assume that an earlier algorithm has constructed the annotated quotient structure $\overline{M}$. In this algorithm, we mark the nodes in $\overline{B}$ whenever we determine that there exists an infinite strong fair path starting from that node and containing infinitely many final nodes. The steps of the algorithm are given below.

1. Construct the automaton $\mathcal{A}$ corresponding to the formula f_i. Construct $\overline{B}$ and B^* from $\overline{M}$ and $\mathcal{A}$. Save $\overline{B}$ as the graph $\overline{H}$.

2. Repeat the following procedure exactly n times where n is the number of processes.

   Compute the MSCCs of $\overline{B}$ and B^*. Determine the final MSCCs in $\overline{B}$. For each final MSCC D of $\overline{B}$ do the following.

   Find all the MSCCs in B^*/D. These will also be MSCCs in B^*. Let $C_1, ..., C_k$ be these MSCCs. If all these MSCCs in B^*/D are strongly fair then mark all the nodes in D. Otherwise do the following. For each j such that C_j is not strongly fair, find all the nodes in C_j that are labeled with en. For each such node $(\overline{x}, j)$, delete $\overline{x}$ from $\overline{B}$, and also delete all nodes of the form $(\overline{x}, k)$ from B^*.

3. Using the graph $\overline{H}$, extend the marking to all nodes that can reach a marked node, i.e. marked in the previous step, by means of an augmeneted path.

4. For each state $\overline{s}$, for some i, if the node $(\overline{s}, init, i)$ is marked then include $\overline{s}$ in the list of nodes that satisfy $\vee_i E_\Phi f_i$.

THEOREM 4.10: The above algorithm correctly determines all the representative states that satisfy the formula $\vee_i E_\Phi f_i$.

Proof: To prove the theorem, it is enough if we show that in the second step of the algorithm, the node $\overline{x}$ is marked iff it belongs to a fair and strongly fair SCC in $\overline{B}$. From all our previous lemmas, it is easy to see that if a node $\overline{x}$ is marked then it must belong to a final and strongly fair SCC in B. To see the other direction, consider the a final and strongly fair SCC, say C, in B that contains $\overline{x}$. Then, there exists a SCC D in $\overline{B}$ such that $C = g(\overline{x}, D)$. It is not difficult to see that in step 2 of the algorithm none of the nodes in D are ever going to be deleted. Now, we show that with in the n iterations in step 2, all nodes in D are going to be marked. Let $\overline{x}$ be a node in D. In any iteration, if the MSCC in B^* containing $(\overline{x}, i)$ is unfair and hence causes the deletion of some nodes, then from the next iteration onwards the MSCC containing $(\overline{x}, i)$ will always be fair, since it will not have any nodes labeled with en. Hence, with in at most n iterations there will be a situation in which, for every $i = 1, ..., n$, the MSCC containing $(\overline{x}, i)$ will be fair. It should be clear that in this iteration all the nodes in the MSCC of $\overline{B}$ that contains $\overline{x}$ (and hence contains all the nodes in D) get marked. $\square$

In order to determine the nodes that satisfy the Fair Indexed CTL* formula $\wedge_i E_\Phi f_i$, we need to modify step 4 of the above algorithm so that we check that, for each $i = 1, ..., n$, $(\overline{s}, init, i)$ is marked.

Assume that the sizes of the annotated structure and that of the automaton $\mathcal{A}$ are m and p, respectively. Here the size of the structure (and also that of the automaton) is the sum of the number states and the number of transitions. Since the number of processes is n, it is easy to see that the size of the structure $\overline{B}$ is mpn, the size of the graph B^* is mpn^2. It is not difficult to see that steps 1,3 and 4 each can be implemented in time $O(mpn)$. Each iteration of step 2 can be implemented in time $O(mpn^2)$ and the n iterations take time $O(mpn^3)$. Thus, the complexity of the above algorithm is $O(mpn^3)$. By recursive descent, we get an algorithm of complexity $O(mp'n^3)$ for the entire Fair Indexed CTL*, where p' is the size of the largest automaton for any basic modality in the input formula.

By the "Litchtenstein-Pnueli Thesis" [16] it is the polynomial (linear) complexity in $|\overline{M}| = m$ that is probably most important for applications. The potential exponential size p of the automaton for f_i (or $g_{i,j}$ or any formula with a bounded number of indices) is likely to be less problematic since specifications are likely to be short while structures may be immense and their quotients still large. If f_i corresponds to a basic modality of Fair Indexed CTL, the automaton is of constant size. Most importantly, we have reduced the complexity in n, the number of processes, to a polynomial factor. (Were it exponential, we would be back to the state explosion problem again.)

To check for correctness under weak (or unconditional) fairness, we can consider it to be a special case of strong fairness and use the above algorithm. However, we can simplify the above algorithm as follows. Call a MSCC in B^* to be weakly fair iff it contains a node that is labeled with ex or that is not labeled with en. Similarly, define a SCC in B to be weakly fair, if for each $i = 1, ..., n$ it contains a node that satisfies the propositional formula $ex_i \vee \neg en_i$. We can reprove all the previous results using these definitions. We modify the above algorithm as follows. In step 2, we simply check if each $C_1, ...C_k$ is weakly fair and if so mark all the nodes in D. We do not need the mutiple iterations in this step. It is easy to see that this modification gives us an algorithm for checking correctness under weak (or unconditional) fairness. Clearly, the complexity of this modified algorithm is $O(mpn^2)$. We also remark that the fair state problem (Given $\overline{M}$ and Φ, does $M, s \models E\Phi$?) is NLOG-space-complete for weak fairness since we can (carefully) guess a "fair" path through $\overline{M}$; it is P-complete for strong fairness.

5 Conclusions

We have shown how to exploit symmetry efficiently under fairness assumptions. It is interesting to note that this could not be done using the essentially group-theoretic approaches of [5, 9], yet works out nicely in the automata-theoretic framework. This seems to testify to the power of automata [20, 21].

One point we have not addressed is the construction $\overline{M}$ incrementally from the program text. This is discussed in [9, 5, 12]. It turns out to reduce to testing if two states are equivalent modulo the group $\mathcal{G} \subseteq Aut\ M$. This equivalence check is, in general, a difficult problem [5]; but when M exhibits many common patterns of symmetry such as full symmetry or rotational symmetry the test for equivalence is particularly easy.

Another point is the relation of our approach of collapsing M according to its symmetry to that of computing $\hat{M}$, the quotient of M modulo its coarsest bisimulation. In principle, this approach should provide the greatest compression, yielding a quotient structure that is possibly smaller than $\overline{M}$[8]. However, there are several unsettled points regarding this latter approach. First, the precise definition of "the" coarsest bisimulation in this context must be given. An additional point is that the coarsest bisimulation must be computed incrementally and it appears problematic to come up with a general, scheme for symbolically representing equivalence classes. In [2] an approach is given that works well on the given examples, but there is no indication of how well it works in general. In [15] a provably fast method of computing coarsest bisimulations is given, with the complexity being measured as a function of the output size – and assuming the existence of certain "oracles" for manipulating equivalence classes. Given that it is not known how to implement such oracles, we really do not know how to efficiently construct quotients modulo the coarsest bisimulation, and we apparently do not know how well such procedures work in practice. Thus we believe that it is an open question as to whether coarsest bisimulation quotients or symmetry quotients are preferable. Certainly, in many practical cases the symmetry quotients can be computed very fast, and it may prove advantageous that any representative state can serve to symbolically represent the entire $\mathcal{G}$-equivalence class.

Finally, it should be noted that our technique for handling fairness could be used in conjunction with the BDD-based implementation framework of [5].

References

[1] 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), North-Holland 1983, 19-34.

[2] Bouajjani, A., Fernandez, J, Halbwichs, N., Raymond, P., and Ratel, C., Minimal State Graph Generation, *Science of Computer Programming*, 1992.

[3] Clarke, E. M., and Emerson, E. A., Design and Verification of Synchronization Skeletons using Branching Time Temporal Logic, Logics of Programs Workshop 1981, Springer LNCS no. 131.

[4] Clarke, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite State Concurrent Programs using Temporal Logic: A Practical Approach, *ACM TOPLAS*, April 1986

[5] Clarke, E. M., Filkorn, T., Jha, S. Exploiting Symmetry in Temporal Logic Model Checking, 5th International Conference on Computer Aided Verification, Crete, Greece, June 1993.

[6] Clarke, E. M., Grumberg, O., and Brown, M., Characterizing Kripke Structures in Temporal Logic, Theor. Comp. Sci., 1988

[8] If $\hat{M}$ is similar to $\overline{M}$, i.e. all transitions are labeled with permutations so that M can be obtained by unwinding $\hat{M}$ according to these permutations, then the algorithms given in this paper can be used for modelchecking using $\hat{M}$ also

[7] Clarke, E. M., Grumberg, O., and Brown, M., Reasoning about Many Identical Processes, Inform. and Comp., 1989

[8] Emerson, E. A. and Lei, C.-L., Modalities for Model Checking: branching Time Strikes Back, *Science of Computer Programming*, v. 8, pp. 275-306, 1987

[9] Emerson, E. A., and Sistla, A. P., Symmetry and Model Checking, 5th International Conference on Computer Aided Verification, Crete, Greece, June 1993; full version to appear in the Journal "Formal Methods in System Design".

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

[11] German, S. M. and Sistla, A. P. Reasoning about Systems with many Processes, Journal of the ACM, July 1992, Vol 39, No 3, pp 675-735.

[12] Ip, C-W. N., Dill, D. L., Better Verification through Symmetry, CHDL, April 1993.

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

[14] 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.

[15] Lee. D., and Yannakakis, M., On-Line Minimization of Transition Systems, STOC92.

[16] Litchtenstein, O., and Pnueli, A., Checking That Finite State Concurrent Programs Satisfy Their Linear Specifications, POPL85

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

[18] Sistla, A. P. and German, S. M., Reasoning with many Processes, Proceedings of the Syposium on Logic in Computer Science, Ithaca, NewYork, 1987

[19] Stirling, C., Modal and Temporal Logics. in Handbook of Logic in Computer Science, (D. Gabbay, ed.) Oxford, 1993

[20] Vardi, M., and Wolper, P., An Automata-Theoretic Framework for Modal Logics of Programs, STOC84

[21] Vardi, M., and Wolper, P., An Automata-Theoretic Framework for Automatic Program Verification, LICS86.