Adaptive Symmetry Reduction[⋆]

Thomas Wahl

Department of Computer Sciences, The University of Texas at Austin, USA
wahl@cs.utexas.edu

Abstract. Symmetry reduction is a technique to counter state explosion for systems of regular structure. It relies on idealistic assumptions about *indistinguishable* components, which in practice may only be *similar*. In this paper we present a generalized algebraic approach to symmetry reduction for exploring a structure without any prior knowledge about its global symmetry. The more behavior is shared among the components, the more compression takes effect. Our idea is to annotate each encountered state with information about how symmetry is violated along the path leading to it. Previous solutions only allow specific types of asymmetry, such as up to bisimilarity, or seem to incur large overhead before or during the verification run. In contrast, our method appeals through its balance between generality and simplicity. We include analytic and experimental results to document its efficiency.

1 Introduction

Symmetry reduction is a well-investigated technique to reduce the impact of state-explosion in temporal logic model checking [2,5]. It has been applied mainly to models of concurrent systems of processes, such as communication and memory consistency protocols. In an ideal scenario, symmetry reduction makes it possible to verify a model over a reduced quotient model, which is not only much smaller, but also bisimulation-equivalent to the original.

The aforementioned ideal scenario is characterized by a transition relation that is invariant under any interchange of the components. In other words, consistently renaming components in both source and target state of any transition must again yield a valid transition in the structure. This condition can be formally violated by systems that nevertheless seem to be approximately symmetric. For example, consider a perfectly symmetric system that evolves into an asymmetric one simply by customizations on some components. The number of transitions that would have to be added or removed in order to make it symmetric is small compared with the total number of transitions.

In this paper we present a new approach to verifying systems of processes with similar behavior. Intuitively, similarity can be expected if many transitions of the system remain valid under many permutations of the processes. Our approach is to annotate each state, space-efficiently, with information about whether and

[⋆] This work is supported in part by NSF grants CCR-009-8141, ITR-CCR-020-5483.

© Springer-Verlag Berlin Heidelberg 2007

how symmetry is violated along the path to it. More precisely, the annotation is a *partition* of the set of all component indices: if the path to the state contains a transition that distinguishes two components, their indices are put into different partition cells. Only components in the same cell can be permuted during future explorations from the state—the algorithm *adapts* to the state's history.

Suppose a given state can be reached along two paths: one with many asymmetric transitions and one with only symmetric ones. This state thus appears twice, once annotated with a fine partition, once with a coarse one. In order to analyze the state's future, we can assume that we reached it along the symmetric path and thus take full advantage of symmetry. The annotated state with the fine partition can be ignored; we say it is *subsumed* by the other one. Subsumption allows us to collapse many states during the exploration. The price we have to pay is that the adaptive algorithm, by its own means, is only suitable for reachability analysis. Throwing away a state subsumed by another leads to an implicit reduced structure that is not bisimulation-equivalent to the original. This price is worth paying since it allows us to improve the analysis of systems with respect to safety properties, a significant and frequent type of formula.

We present an exact and efficient algorithm for reachability analysis, suitable especially on an *approximately fully symmetric* Kripke structure. The property to be verified may be asymmetric, i.e. it may distinguish between components. Errors are discovered at minimum distance from the initial state, and paths to them can be recovered, provided a breadth-first search order. Following the presentation of the technical details of our method, we give analytic and practical results substantiating its usefulness.

2 Related Work

There are many publications on the use of symmetry for state space exploration and model checking, both of fundamental nature [2,5] and specific for tools [7,8]. One of the first to apply symmetry reduction strategies to partially symmetric systems is [6]. The authors present the notions of *near* and *rough* symmetry, which are defined with respect to a Kripke structure; especially for rough symmetry it is unclear how to verify it on a high-level system description. Examples are limited to versions of the *Readers-Writers* problem.

This work was generalized in [4] to *virtual symmetry*, the most general condition that allows a bisimilar symmetry quotient. A limitation of all preceding approaches is the existence of a strict precondition for their principle applicability. As with [6], it is left open whether virtual symmetry can be verified efficiently; the techniques presented in [4] seem to incur a cost proportional to the size of the unreduced Kripke structure. On the other hand, bisimilarity makes these approaches suitable for full μ-calculus model checking, whereas the adaptive technique trades "property coverage" in for "system coverage".

Symmetry detection solves the problem of suspected but formally unknown symmetry by inferring structure automorphisms from the program text [3]. This approach is principally different from ours. A structure automorphism is global in character, being defined over the transition relation. It ignores the possibility of a large part of the state space being unaffected by symmetry breaches. The adaptive approach, which can be viewed as *on-the-fly symmetry violation detection*, operates directly on the Kripke structure. As such, it can reduce local substructures with more symmetry than revealed by global automorphisms.

Closest in spirit to our work is that by P. Sistla and P. Godefroid [9], who also target arbitrary systems and properties. A *guarded annotated quotient* is obtained from a symmetric super-structure by marking transitions that were added to achieve symmetry. As an advantage, this method can handle arbitrary CTL* properties. In our work, annotations apply to states, not edges, and seem more space-efficient; in [9] there can be multiple annotations to a quotient edge. Further, the adaptive method does not require any preprocessing of the program text, such as in order to determine a symmetric super-structure.

3 An Example

Consider the variant of the *Readers-Writers problem* shown in figure 1. There are two "reader" processes (indices 1, 2) and one "writer" (3). In order to access some data item, each process must enter its critical section, denoted by local state C. The edge from (the non-critical section) N to (the trying region) T is unrestricted, as is the one from C back to N. There are two edges from T to C.

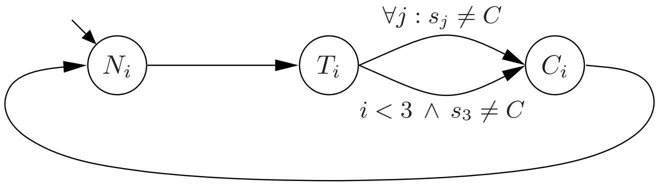

Fig. 1. Local state transition diagram of process i for an asymmetric system

The first is executable whenever no process is currently in its critical section ($\forall j : s_j \neq C$, for current state s). The second is available only to readers ($i < 3$), and the writer must be in a non-critical local state ($s_3 \neq C$). Intuitively, since readers only read, they may enter their critical section at the same time, as long as the writer is outside its own.

With each process starting out in local state N, the induced Kripke structure has 22 reachable states. The adaptive method, however, constructs a reachability tree of only 9 *abstract* states (figure 2).An abstract state of the form XYZ represents the set of concrete states obtained by permuting the local state tuple (X, Y, Z). Consider, for example, the abstract state NNT, representing (N, N, T), (N, T, N) and (T, N, N). Guard $\forall j : s_j \neq C$ of the first edge from T to C is satisfied in all three states. Executing this edge leads to the successor states (N, N, C), (N, C, N), (C, N, N), succinctly written as NNC in figure 2.

$$\begin{tikzcd}
 & & NNC \arrow[r] & NTC \arrow[r] & CC \mid N \arrow[r] & CC \mid T \\
NNN \arrow[r] & NNT \arrow[ur] \arrow[dr] & & & & \\
 & & NTT \arrow[r] & TTT \arrow[r] & TTC &
\end{tikzcd}$$

Fig. 2. Abstract reachability tree for the model induced by figure 1

Now consider the abstract state NTC. None of the six concrete states it represents satisfies the condition $\forall j : s_j \neq C$. Thus, regarding steps from T to C, we have to look at the second—asymmetric—edge, guarded by $i < 3 \wedge s_3 \neq C$. Of the six represented states, two satisfy this condition with an index $i < 3$ such that $s_i = T$, namely (T, C, N) and (C, T, N). In both cases, the edge leads to state (C, C, N). We now have to make a note that this state is reached through an asymmetric edge. The edge's guard is invariant under the transposition (1 2), but not under any permutation displacing index 3. We express this succinctly in figure 2 as abstract state $CC \mid N$. Intuitively, permutations across the "|" are illegal; this abstract state hence represents neither (N, C, C) nor (C, N, C).

We finally remark that the induced structure is not *virtually* symmetric and hence not *nearly* or *roughly* so. To see this, consider the (valid) transition $(T, C, T) \to (C, C, T)$. Applying transposition (2 3) to it we obtain transition $(T, T, C) \to (C, T, C)$, which is invalid, but belongs to the structure's *symmetrization* [4]. Virtual symmetry requires a way to permute the target state that makes the transition valid, which is impossible here. As a corollary, this structure is not bisimilar to its natural symmetry quotient.

4 Preliminaries: Permutations, Symmetry, Partitions

Consider a Kripke structure $M = (S, R)$ modeling a system of n concurrently executing processes. Let Sym_n be the group of *permutations* on $[1..n]$ and let $\pi \in Sym_n$ operate on a state $s \in S$ in the form $\pi(s_1, \ldots, s_n) = (s_{\pi(1)}, \ldots, s_{\pi(n)})$. M is said to be *fully symmetric* if for every $\pi \in Sym_n$,

$$(s, t) \in R \quad \text{iff} \quad (\pi(s), \pi(t)) \in R\,. \tag{1}$$

A symmetric structure can be reduced to a bisimilar and smaller quotient structure based on the *orbit relation*: $s \equiv t$ iff $\exists \pi : \pi(s) = t$. More details of symmetry reduction are available in the literature [2,5].

A *partition* of $[1..n]$ is a set of disjoint, non-empty subsets, called *cells*, that cover $[1..n]$. We use a notation of the form $\mid 1,4 \mid 2,5 \mid 3,6 \mid$ to represent the partition into the three cells $\{1,4\}$, $\{2,5\}$ and $\{3,6\}$. The coarsest partition $\mid 1, \ldots, n \mid$ consists of a single cell, the finest partition $\mid 1 \mid \ldots \mid n \mid$ consists of n singleton cells. A partition $\mathbb{P}$ induces an equivalence relation on $[1..n]$: we write $i \equiv_{\mathbb{P}} j$ exactly if i and j belong to the same cell of $\mathbb{P}$.

We say a partition $\mathbb{P}$ of $[1..n]$ *generates* all permutations π on $[1..n]$ such that for all i, $i \equiv_{\mathbb{P}} \pi(i)$. These permutations form a group, denoted by $\langle \mathbb{P} \rangle$. For example, the partition $\mid 1,4 \mid 2,5 \mid 3,6 \mid$ generates a group of six permutations.

The coarsest partition $|\,1,\dots,n\,|$ generates the entire symmetry group Sym_n. The finest partition $|\,1\,|\dots|\,n\,|$ generates only the identity permutation.

5 Computational Model

We assume a system is modeled as a local state transition diagram. This level of abstraction is fully expressive for shared-memory systems and lets us focus on synchronization aspects. Precisely, the system is specified as a number n of processes and a graph with local states as nodes. Local transitions, called *edges*, have the form

$$A \xrightarrow{\phi,\mathbb{Q}} B\,. \tag{2}$$

ϕ is a two-place predicate taking a state s and an index i. State s defines the *context* in which the edge is to be executed [1]. The intended semantics is that $\phi(s,i)$ returns *true* exactly if in state s process i is allowed to transit from local state A to local state B. Predicate ϕ can be written in any efficiently decidable logic, such as propositional logic with simple arithmetic over state variables and index i. In figure 1 we have seen the predicate

$$\phi(s,i) \;\; = \;\; i<3 \;\wedge\; s_3 \neq C\,. \tag{3}$$

It is asymmetric (and thus is the edge) since we can find s, i and a permutation π such that $\phi(s,i) \neq \phi(\pi(s),\pi(i))$. On the other hand, asymmetric edges are often symmetric with respect to a subgroup of Sym_n. For instance, predicate (3) is invariant under the transposition $\sigma = (1\ 2)$, i.e. $\phi(s,i) = \phi(\sigma(s),\sigma(i))$ for all s, i. In common variants of the r-readers/$(n-r)$-writers problem, the asymmetric edges are immune to any products of permutations of $[1..r]$ and $[r{+}1..n]$. Such permutations are generated by the partition $|\,1..r\,|\,r{+}1..n\,|$.

Symbol $\mathbb{Q}$ in equation (2) stands for a partition generating the automorphism group of the edge, i.e. a set of permutations that preserve predicate ϕ. For the asymmetric edge in (3), we choose $\mathbb{Q} = |\,1,2\,|\,3\,|$. In approximately symmetric systems, $\mathbb{Q}$ is for most edges the coarsest partition, generating Sym_n. For the remaining edges—those that destroy the symmetry—we expect the user to provide a suitable $\mathbb{Q}$. The high-level description of the edge often suggests a group of automorphisms; see section 9 for an example. If needed, a propositional SAT-solver can aid the verification of the automorphism property.

Letting l be the number of local states, an asynchronous semantics of the induced n-process concurrent system is given by the following Kripke structure: $S := [1..l]^n$, and R is the set of transitions $(s_1,\dots,s_n) \to (t_1,\dots,t_n)$ with the property that there is an index $i \in [1..n]$ such that

1. there exists an edge $s_i \xrightarrow{\phi,\mathbb{Q}} t_i$ with $\phi((s_1,\dots,s_n),i) = true$ and
2. $\forall j : j \neq i : s_j = t_j$.

Note that $\mathbb{Q}$ is irrelevant for the definition of the Kripke structure; it is instead part of the syntax of an edge. Extending the method to work with shared variables or with a synchronous execution semantics is fairly straightforward.

6 Orbits and Subsumption

The goal of this paper is an efficient exploration algorithm for the Kripke structure defined in the previous section. The algorithm accumulates states annotated with partitions that indicate how symmetry was violated in reaching this state. Thus, the formal search space of the exploration is the set $\hat{S} := [1..l]^n \times Part_n$, where $Part_n$ is the set of all partitions of $[1..n]$. The partition is used to determine which permutations can be applied to the state in order to obtain the concrete states it represents. These permutations are those that do not permute elements across cells, i.e. those generated by the partition (see end of section 4):

Definition 1. *Let π be a permutation on $[1..n]$. For an n-tuple $s = (s_1, \ldots, s_n)$, let $\pi(s)$ denote the expression $(s_{\pi(1)}, \ldots, s_{\pi(n)})$. We extend π to operate on an element $\hat{s} = (s, \mathbb{P})$ of $\hat{S}$ in the form*

$$\pi(s, \mathbb{P}) = \begin{cases} (\pi(s), \mathbb{P}) & \textit{if } \pi \in \langle \mathbb{P} \rangle \\ (s, \mathbb{P}) & \textit{otherwise.} \end{cases}$$

This mapping defines a bijection on $\hat{S}$. Note that π never changes the partition associated with a state; if π is not generated by $\mathbb{P}$, it does not affect $(s, \mathbb{P})$ at all.

In standard symmetry reduction, algorithms operate on representative states of orbit equivalence classes. Systems with asymmetries require a generalized notion of an orbit that defines the relationship between states in $\hat{S}$ and in S:

Definition 2. *The* orbit *of a state $\hat{s} = (s, \mathbb{P}) \in \hat{S}$ is defined as*

$$orbit(s, \mathbb{P}) = \{t \in S : \exists \pi \in \langle \mathbb{P} \rangle : \pi(s) = t\} .$$

We say that $\hat{s}$ represents *t if $t \in orbit(\hat{s})$.*

Examples. For $n = 4$, consider the following states and the sizes of their orbits:

$\hat{s} = (s, \mathbb{P})$	orbit size
$(ABCD,\ \lvert 1,2,3,4 \rvert)$	$4! = 24$ (standard symmetry)
$(ABCD,\ \lvert 1,2 \mid 3,4 \rvert)$	$2 \times 2 = 4$
$(ABCD,\ \lvert 1,2 \mid 3 \mid 4 \rvert)$	$2 \times 1 \times 1 = 2$
$(ABCD,\ \lvert 1 \mid 2 \mid 3 \mid 4 \rvert)$	$1 \times 1 \times 1 \times 1 = 1$

If $\mathbb{P}$ is the coarsest partition $\lvert 1, \ldots, n \rvert$, then $orbit(s, \mathbb{P})$ reduces to the equivalence class that s belongs to under the standard orbit relation.

Subsumption. Orbits in standard symmetry reduction are equivalence classes and as such either disjoint or equal. In contrast, the new orbit definition is not based on an equivalence relation. Indeed, the orbits of the four example states in the table above form a strictly descending chain. It is therefore unnecessary to remember all four states if encountered during exploration: the first *subsumes* the others.

Definition 3. *State $\hat{s} \in \hat{S}$* subsumes *$\hat{t} \in \hat{S}$, written $\hat{s} \rhd \hat{t}$, if $orbit(\hat{s}) \supseteq orbit(\hat{t})$.*

Examples. For $n = 3$, consider the following states and examples of what they subsume and don't subsume ($\mathbb{Q}$ is arbitrary):

$\hat{s} = (s, \mathbb{P})$	$\hat{s}$ subsumes:	$\hat{s}$ does not subsume:
$(ABC,\ \lvert 1,2,3 \rvert)$	$(ABC,\ \mathbb{Q})$, $(BCA,\ \mathbb{Q})$	$(ABB,\ \mathbb{Q})$
$(ABC,\ \lvert 1,3 \lvert 2 \rvert)$	$(ABC,\ \lvert 1 \lvert 2 \lvert 3 \rvert)$, $(CBA,\ \lvert 1 \lvert 2 \lvert 3 \rvert)$	$(BAC,\ \mathbb{Q})$
$(ABC,\ \lvert 1 \lvert 2 \lvert 3 \rvert)$	itself only	$(ABC,\ \lvert 1,3 \lvert 2 \rvert)$

Definition 3 provides no clue about how to efficiently detect subsumption. An alternative characterization is the following. Recall that $i \equiv_{\mathbb{P}} j$ iff i and j belong to the same cell within $\mathbb{P}$.

Theorem 4. *State* $\hat{s} = (s, \mathbb{P})$ *subsumes state* $\hat{t} = (t, \mathbb{Q})$ *exactly if*

1. $i \equiv_{\mathbb{Q}} j \ \Rightarrow\ (i \equiv_{\mathbb{P}} j \vee t_i = t_j)$ *is a tautology, and*
2. $t \in \mathit{orbit}(\hat{s})$.

Remark. Condition 1 is slightly weaker than the condition $i \equiv_{\mathbb{Q}} j \Rightarrow i \equiv_{\mathbb{P}} j$, which states that $\mathbb{P}$ is coarser than $\mathbb{Q}$. As a hint why $t_i = t_j$ is needed for an equivalent characterization of subsumption, consider $\hat{s} = (AA, \lvert 1 \lvert 2 \rvert)$, which has a finer partition than $\hat{t} = (AA, \lvert 1, 2 \rvert)$, but subsumes $\hat{t}$.

Condition 1 can, using appropriate data structures for partitions, be decided in $\mathcal{O}(n^2)$ time. In practice, violations are often detected much faster using heuristics such as comparing the cardinalities of $\mathbb{P}$ and $\mathbb{Q}$. Condition 2 requires checking whether $\mathbb{P}$ generates a permutation π that satisfies $\pi(s) = t$. This can be decided in $\mathcal{O}(n)$ time by treating each cell $P \in \mathbb{P}$ separately: we project both s and t to the positions in P and use a counting argument to verify that the projections are the same up to permutation.

Algebraic Properties of Subsumption. Relation $\triangleright$ is a *preorder*: it is reflexive and transitive. It is, however, neither symmetric (e.g. $(AB, \lvert 1, 2 \rvert) \triangleright (AB, \lvert 1 \lvert 2 \rvert)$ but not vice versa) nor anti-symmetric (e.g. $(AB, \lvert 1, 2 \rvert)$ and $(BA, \lvert 1, 2 \rvert)$ subsume each other but differ). Thus, it is neither an equivalence nor a partial order.

We can derive an equivalence relation from a preorder by making it bidirectional: write $\hat{s} \bowtie \hat{t}$ if $\hat{s} \triangleright \hat{t} \wedge \hat{t} \triangleright \hat{s}$. How is this equivalence related to the *orbit relation* on $\hat{S}$, written $\hat{s} \equiv \hat{t}$ if there exists π such that $\pi(\hat{s}) = \hat{t}$?

Lemma 5. *For any* $\hat{s}, \hat{t} \in \hat{S}$, $\hat{s} \equiv \hat{t}$ *implies* $\hat{s} \bowtie \hat{t}$.

According to the lemma, the orbit relation achieves less compression than subsumption: the latter is coarser, i.e. it relates more states. We note that in perfectly symmetric systems, where each state is (implicitly) annotated with the coarsest partition $\lvert 1, \ldots, n \rvert$, the three relations $\triangleright$, $\bowtie$ and $\equiv$ coincide.

7 State Space Exploration Under Partial Symmetry

We are now ready to present an algorithm for state space exploration on the (partially symmetric) structure $M = (S, R)$. The goal is to compute the set of

Algorithm 1. State space exploration under partial symmetry

```
Input: initial state $s_0 \in S$
 1: $\mathit{Reached} := \mathit{Unexplored} := \{(s_0, |\, 1, \ldots, n \,|)\}$
 2: while $\mathit{Unexplored} \neq \emptyset$ do
 3:   let $\hat{s} = (s, \mathbb{P}) \in \mathit{Unexplored}$; remove $\hat{s}$ from $\mathit{Unexplored}$
 4:   for all edges $e = A \xrightarrow{\phi, \mathbb{Q}} B$ do
 5:     $\mathbb{R} := \mathit{glb}(\mathbb{P}, \mathbb{Q})$
 6:     $U := \mathit{unwind}(s, \mathbb{P}, \mathbb{Q})$
 7:     for all states $u \in U$ do
 8:       for all cells $R \in \mathbb{R}$ do
 9:         if $\exists i \in R : u_i = A \;\wedge\; (u, i) \models \phi$ then
10:           $v := (u_1, \ldots, u_{i-1}, B, u_{i+1}, \ldots, u_n)$
11:           $\mathit{canonicalize}(v)$
12:           $\mathit{update}(v, \mathbb{R})$
```

states reachable under R from some initial state $s_0 \in S$. Technically, the algorithm operates on elements of $\hat{S}$; we later present a one-to-one correspondence between the states reachable in M and the states found by algorithm 1.

In line 1, the initial state is annotated with the coarsest partition (indicating absence of symmetry violations so far) and put on the *Unexplored* and *Reached* lists. While available, one state $\hat{s}$ is selected from *Unexplored* for expansion.

Successors of $\hat{s}$ are found by iterating through all edges (line 4). We now have to reconcile two partitions: $\mathbb{P}$, expressing symmetry violations on the path to s, and $\mathbb{Q}$, expressing violations to be caused by e. Routine *glb* in line 5 determines the partition $\mathbb{R}$ such that $\langle \mathbb{R} \rangle = \langle \mathbb{P} \rangle \cap \langle \mathbb{Q} \rangle$. $\mathbb{R}$ can be computed as the greatest lower bound (meet) of $\mathbb{P}$ and $\mathbb{Q}$ in the complete lattice of partitions, which uses "finer-than" as the partial order relation.

Edge predicate ϕ may not be invariant under permutations from $\langle \mathbb{P} \rangle$, but it is under permutations from $\langle \mathbb{Q} \rangle$ and thus from $\langle \mathbb{R} \rangle$. We account for this fact by unwinding s into a set of states to be annotated by $\mathbb{R}$ whose orbits exactly *cover* the orbit of $\hat{s} = (s, \mathbb{P})$, i.e. into a set $U \subset S$ that satisfies

$$\bigcup_{u \in U} \mathit{orbit}(u, \mathbb{R}) = \mathit{orbit}(s, \mathbb{P})\,. \tag{4}$$

The objective is of course to find a *small* set U with this property. In line 6, routine *unwind* returns the set $U = \{s\} \cup \{\pi(s) : \pi \in \langle \mathbb{P} \rangle \setminus \langle \mathbb{Q} \rangle\}$, which is easily seen to satisfy (4). This step can be a bottleneck; we discuss in section 8 how to avoid it in most cases and alleviate it in the remaining ones.

Processes with indices in different cells of $\mathbb{R}$ are distinguishable; we must consider these cells separately (line 8). Edge e can be executed if there is a process i in local state A such that (u, i) satisfies ϕ. If so, we let the process proceed, resulting in a new state v (line 10). In line 11, v is *canonicalized* within R: the sequence of local states with indices in R is lexicographically sorted.

The *update* function determines whether to add a new state $\hat{v}$ to the lists *Unexplored* and *Reached* (algorithm 2). If some state in *Reached* subsumes $\hat{v}$,

Algorithm 2. Updating *Unexplored* and *Reached*: *update*$(v, \mathbb{R})$

```
Input: newly computed state $\hat{v} = (v, \mathbb{R})$
1: if no state in Reached subsumes $\hat{v}$ then
2:     check whether $\hat{v}$ represents a concrete error state
3:     remove from Unexplored each $\hat{w}$ such that $\hat{v} \rhd \hat{w}$
4:     add $\hat{v}$ to Unexplored and to Reached
```

nothing needs to be done; this also covers the case $\hat{v} \in$ *Reached*. Otherwise (line 2), $\hat{v}$ is checked for errors (discussed below). Then, states that $\hat{v}$ subsumes are removed from *Unexplored*: such states are implicitly explored as part of $\hat{v}$ and are thus redundant. Finally, $\hat{v}$ is added to both lists.

States reachable from s_0 in M are related to states in *Reached* as follows.

Theorem 6. *Let $s_0 \in S$ and Reached as computed by algorithm 1. A state $s \in S$ is reachable from s_0 in M exactly if there exists $\hat{s} \in$ Reached that represents s.*

Error conditions to be checked in line 2 of algorithm 2 need not be symmetric. For example, suppose the claim is that process 3 never enters local state X. Given $\hat{v} = (v, \mathbb{R})$, we determine the unique cell $R \in \mathbb{R}$ such that $3 \in R$. An error is reported exactly if the property $\exists i \in R : v_i = X$ evaluates to *true*.

If M has an error at distance d from s_0, then algorithm 1, if organized in a breadth-first fashion, detects it at distance d from the root of the abstract reachability tree. Using back-edges from each encountered node to its predecessor, a shortest error path can be reconstructed and lifted to a concrete path as usual.

Regarding line 3 of algorithm 2, the only reason not to remove $\hat{w}$ from *Reached* (but only from *Unexplored*) is to retain the ability to trace encountered errors back to the initial state, for which those states may be needed. They are not needed for just *finding* errors or for termination detection.

8 Implementation and Efficiency

We discuss essential refinements of algorithm 1 and derive analytic results.

In approximately symmetric systems, most edges are symmetric, resulting in a search that annotates many states with the coarsest partition $|\,1, \ldots, n\,|$. We encode this partition space-efficiently using the empty string. Further, a symmetric edge e in line 4 of algorithm 1 allows dramatic simplifications: Lines 5, 6 and 7 can be removed, as $\mathbb{R}$ equals $\mathbb{P}$ and U reduces to $\{s\}$. The test $(u, i) \models \phi$ can be factored out of the loop in line 8 (replacing i with 0), since it is independent of i (due to ϕ's symmetry). Almost the same simplifications apply if e is asymmetric but $\mathbb{Q}$ is coarser than $\mathbb{P}$ ($\langle\mathbb{Q}\rangle \supseteq \langle\mathbb{P}\rangle$), which is easy to test.

If $\mathbb{Q}$ is finer than $\mathbb{P}$, we must compute $U = \{s\} \cup \{\pi(s) : \pi \in \langle\mathbb{P}\rangle \setminus \langle\mathbb{Q}\rangle\}$. Doing this by enumerating $\langle\mathbb{P}\rangle \setminus \langle\mathbb{Q}\rangle$ is inefficient and unnecessary: state s likely contains redundancy in the form of duplicate local states (especially if there are more processes than local states). Thus, many permutations of $\langle\mathbb{P}\rangle \setminus \langle\mathbb{Q}\rangle$ result in the same state when applied to s. This redundancy can be avoided up front using *buckets*, i.e. sets of process counters for each local state, separately in each cell of $\mathbb{Q}$. Permutations outside $\langle \mathbb{Q} \rangle$ are applied to s by changing the contents of the buckets. As a result, the complexity of *unwind* is proportional to $|U|$, which is usually much smaller than $|\langle \mathbb{P} \rangle \setminus \langle \mathbb{Q} \rangle|$. The set U itself is large only when $\mathbb{Q}$ is very fine, which is not typical for approximately symmetric systems.

To make the *update* function in algorithm 2 efficient, the list *Reached* is sorted such that states with local state vectors that are permutations of each other are adjacent, for example states of the forms $(AAB, \mathbb{P}_1)$, $(AAB, \mathbb{P}_2)$, $(BAA, \mathbb{P}_3)$. Given the newly reached $\hat{v} = (v, \mathbb{R})$, we first use binary search to identify the range in which to look for candidates for subsumption as the *contiguous* range of states in *Reached* whose local state vectors are permutations of v. The search in line 1 of algorithm 2 for states subsuming $\hat{v}$ can now be limited to this range.

We present complexity bounds for the adaptive exploration technique. Consider the abstract state space $\hat{S} = S \times \mathit{Part}_n$, which is conceivably much bigger than S. Our algorithm, however, only explores states not subsumed by others. Comparing the adaptive technique to standard symmetry reduction and to *plain* exploration oblivious to symmetry, our informal goal is to show that

$$\mathit{complexity}(\mathit{adaptive}) \quad \leq \quad \mathit{complexity}(\mathit{standard}) \quad < \quad \mathit{complexity}(\mathit{plain})\,. \tag{5}$$

If the automorphism group of the structure induced by a program is non-trivial, standard symmetry reduction is guaranteed to achieve some compression.[1] The meaning of "$\leq$" in (5) is that this compression is preserved by our technique.

To demonstrate this, we first quantify the effect of standard symmetry reduction on a program in our input syntax. Call two processes *friends* if they are not distinguished by any edge, i.e. for each edge $A \xrightarrow{\phi, \mathbb{Q}} B$ there is a cell $Q \in \mathbb{Q}$ containing both processes. Friendship is an equivalence relation on $[1..n]$. Each class of friends induces a group of permutations that can be extended to automorphisms of the program's Kripke structure. The orthogonal product of these groups is the largest symmetry group that can be derived from the program text.

Friends enjoy the following property:

Theorem 7. *Let F be a set of friends. Algorithm 1 reaches at most $\binom{|F|+l-1}{|F|}$ local state tuples over the indices in F.*

The quantity in theorem 7 equals the number of representative states under standard symmetry reduction over the group $\mathit{Sym}\, F$ of all permutations of F. As a special case, if all n processes are friends, algorithm 1 reduces to standard symmetry reduction and introduces nearly no search overhead.

Whether the "$\leq$" in (5) is actually "$<$" or "$\ll$" depends on the way symmetry is violated and is hard to quantify analytically. We observe, however, that for the adaptive technique, the notion of *friends* can be extended to include processes not distinguished by edges that are actually *followed* during the exploration.

[1] We overlook the pathological case in which only symmetric states are reachable.

Unreachable asymmetric edges reduce the automorphism group, but have no effect on our algorithm. This observation is supported by our experimental results.

9 Experimental Evaluation

We tested the adaptive method in a variety of experiments. We borrow a resource controller example from the work by Sistla and Godefroid [9, p. 729ff.]. In short, process indices are partitioned into intervals of equal priority. In case of simultaneous requests, a server grants the resource to one of the highest-priority processes, thus introducing asymmetry. For a process belonging to the priority interval $[l_c..u_c]$, we annotate each asymmetric edge with the partition $|\, 1,\ldots,l_c{-}1 \,|\, l_c,\ldots,u_c \,|\, u_c{+}1,\ldots,n \,|$, separating higher, equal and lower priority.

In a first set of experiments, we compare the memory use of the adaptive technique to *plain* exploration oblivious of symmetry. Memory is measured by the (reproducible) number of reached states (memory in bytes is linear in this number, including the overhead due to the annotations). Figure 3 plots this

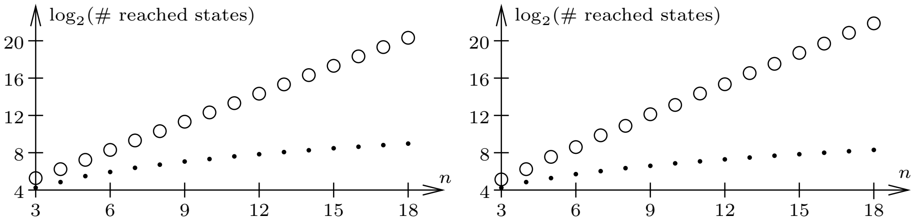

Fig. 3. Comparing the adaptive technique (small dots) to plain exploration (large circles): reached states for $n/2$ small priority classes (left) and two large classes (right)

number over various process counts n for the adaptive technique (small dots) and plain exploration (large circles) on a logarithmic scale. The graphs on the left and on the right differ in the priority scheme used. For $n = 18$, the plain algorithm reaches $1,310,716$ states on the left and $3,808,000$ on the right, whereas our algorithm reaches only 505 abstract states on the left and 316 on the right. The right scheme allows more compression due to larger priority classes; the 316 abstract states reached by our algorithm very compactly represent the $3,808,000$ concrete ones. In all cases, the adaptive algorithm took nearly zero time; for the plain algorithm the largest time measured is 7:16min.

In a second set of experiments, we compare the memory use of the adaptive technique with standard symmetry reduction, based on the induced structure's automorphism group (figure 4). For the highly fragmented scheme on the left, the standard algorithm does quite poorly (thus again the logarithmic scale): for $n = 18$, it reaches $78,729$ states, compared with 505 adaptively. The maximum symmetry group is the product of the 9 transpositions (1 2) through (17 18),

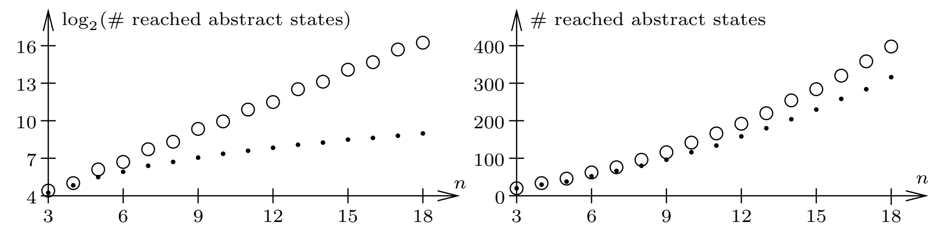

Fig. 4. Comparing the adaptive technique (small dots) to standard symmetry reduction (large circles); priority schemes as in figure 3

yielding a group size and expected compression factor of only $2^9 = 512$. This effect is much less severe for the less fragmented scheme on the right (linear scale), as is clearly revealed by the graph.

In a third set of experiments, we directly investigate how the adaptive method scales with increasing fragmentation; the idea to do this is again borrowed from [9]. The resource controller example with k priority classes is run with a large number of 80 processes. The objective is to look for states where a process holds the resource while the resource is globally recorded to be free. In a first variant, denoted "$1, 1, \ldots, rest$", all priority classes but the last contain a single process; the last contains the rest. In a second variant, denoted "$2, 2, \ldots, rest$", all classes but the last contain two processes; the last contains the rest. We see from table 1 that the number of reached states grows roughly linearly with k;

Table 1. Adaptive symmetry reduction against increasing fragmentation

k	n	"$1, 1, \ldots, rest$"		"$2, 2, \ldots, rest$"	
		Time	# states	Time	# states
2	80	1s	558	1s	789
3	80	2s	792	4s	1245
5	80	4s	1251	13s	2121
7	80	8s	1698	24s	2949

k	n	"$1, 1, \ldots, rest$"		"$2, 2, \ldots, rest$"	
		Time	# states	Time	# states
10	80	14s	2346	45s	4101
15	80	28s	3366	83s	5781
20	80	44s	4311	118s	7161
25	80	62s	5181	151s	8241

computation times are very reasonable. For fixed k, the fragmentation grows with increasing size of the initial k classes (1 vs. 2), since then the final class (hosting the majority of the processes) becomes smaller.

For $k \leq 5$, data obtained with the GQS-based method were provided in [9]. Those running times are an order of magnitude higher, although they of course depend on the machine used. Reproducible memory data for these examples (such as the number of reached states) were not given in [9].

10 Summary

We presented a new *adaptive* method for exhaustive state space exploration. It is intended for, and efficient with, *approximately fully symmetric* systems, where many transitions are shared by most processes. Verification of this feature is not required; the method is exact for any input. We introduced the notion of *subsumption*: a state subsumes another if its orbit contains that of the other one. Subsumption induces a quotient structure with an identical set of reachable states. We focused on full symmetry, since this type is the most frequent and profitable in practice. The adaptive method can be implemented as well for rotation groups; critical is the ability to represent and manipulate groups succinctly. Our implementation uses an explicit state representation. We believe the algorithm can be incorporated into the Murφ model checker [8] and extend its applicability to asymmetric systems.

The subsumption relation benefits reachability analysis by aggressively suppressing re-emerging states, even non-equivalent ones. This behavior is too crude for general model checking; how to extend the method is part of our future work.

References

1. Abdulla, P.A., Bouajjani, A., Jonsson, B., Nilsson, M.: Handling global conditions in parameterized system verification. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, Springer, Heidelberg (1999)
2. Clarke, E.M., Enders, R., Filkorn, T., Jha, S.: Exploiting symmetry in temporal logic model checking. In: Formal Methods in System Design (FMSD) (1996)
3. Donaldson, A.F., Miller, A.: Automatic symmetry detection for model checking using computational group theory. In: Fitzgerald, J.A., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, Springer, Heidelberg (2005)
4. Emerson, E.A., Havlicek, J.W., Trefler, R.J.: Virtual symmetry reduction. In: Logic in Computer Science (LICS) (2000)
5. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Formal Methods in System Design (FMSD) (1996)
6. Emerson, E.A., Trefler, R.J.: From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, Springer, Heidelberg (1999)
7. Hendriks, M., Behrmann, G., Larsen, K.G., Niebert, P., Vaandrager, F.W.: Adding symmetry reduction to Uppaal. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, Springer, Heidelberg (2004)
8. Ip, C.N., Dill, D.L.: Verifying systems with replicated components in *mur*ϕ. Formal Methods in System Design (FMSD) (1999)
9. Sistla, A.P., Godefroid, P.: Symmetry and reduced symmetry in model checking. ACM Trans. on Programming Languages and Systems (TOPLAS) (2004)