Appeared in Formal Aspects of Computing

A Lazy Approach to Symmetry Reduction

Thomas Wahl and Vijay D'Silva
Computing Laboratory, University of Oxford, United Kingdom

Keywords: Model Checking; Symmetry; Annotated Structure; Partition; Subsumption

Abstract. *Symmetry reduction* is a technique to counter state explosion for systems with regular structure. It relies on idealistic assumptions about indistinguishable components, which in practice may only be similar. In this article, we present a flexible, lazy approach to symmetry-reducing a structure without any prior knowledge about its global symmetry. Instead of a-priori checking for compliance with symmetry conditions, each encountered state is annotated on the fly with information about how symmetry is *violated* along the path leading to it. The method naturally favors "very symmetric" systems: more similarity among the components leads to greater compression. A notion of *subsumption* is used to prune the annotated search space during exploration. Previous solutions to the approximate symmetry reduction problem are restricted to specific types of asymmetry, such as up to bisimilarity, or incur a large overhead, either during preprocessing of the structure or during the verification run. In contrast, the strength of our method is its balance between ease of implementation and algorithmic flexibility. We include analytic and experimental results that witness its efficiency.

1. Introduction

Symmetry reduction is a well-investigated technique to curb the impact of state explosion on temporal logic model checking. It has been applied mainly to domain-specific abstract 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, 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 yields again a valid transition in the structure. This condition can be formally violated by systems that nevertheless seem to be approximately symmetric. For example, consider an initially perfectly

Correspondence and offprint requests to: Thomas Wahl, Computing Laboratory, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom, email: `wahl@comlab.ox.ac.uk`. A preliminary version of this article appeared in [Wah07]. The work was supported by the EPSRC, grant number EP/G026254/1, and by a Microsoft European PhD Fellowship.

symmetric system that evolves into an asymmetric one through customizations on some components. The number of transitions of the asymmetric system that would have to be modified in order to make the system symmetric is small compared with the total number of transitions. There is thus an incentive to exploit the structure of the asymmetric system.

In this article 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 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 appear in different partition cells. An annotated state $(s, \mathbb{P})$ functions as a representative for all states that can be obtained from s by applying a permutation *generated* by $\mathbb{P}$: only components in the same $\mathbb{P}$-cell may be permuted. The symmetry exploitable during future explorations from s is thus restricted by $\mathbb{P}$; the algorithm is in this sense adaptive.

Suppose a state s can be reached along two paths: one with only symmetric transitions, giving rise to a coarse partition $\mathbb{P}$, and one with many asymmetric transitions, giving rise to a finer partition $\mathbb{Q}$. We now observe that every state represented by $(s, \mathbb{Q})$ is also represented by $(s, \mathbb{P})$. When analyzing the future of s, we can therefore discard the inconveniently fragmented state $(s, \mathbb{Q})$; we say it is *subsumed* by $(s, \mathbb{P})$. Subsumption allows us to collapse many states during the exploration and results in an implicit reduced structure that is *shortest-path equivalent* to the original. The price we have to pay for the reduction is that the algorithm is only suitable for reachability analysis. As we demonstrate in section 10, the significant speedups obtained by the reduction make the price worth paying.

Our proposed technique can be viewed as *lazy symmetry reduction*: it postpones worrying about the precision of the reduction until symmetry violations actually occur. Repeatedly adjusting the set of applicable permutations appears more expensive than detecting symmetry up front on the program text, and it is. The advantages of delaying the detection to the model checking run are that (i) unreachable parts of the system's Kripke structure have no impact on the exploitable symmetry, and (ii) symmetric *substructures* can be reduced locally. Standard symmetry reduction imposes an unreasonable penalty on systems where strict symmetry is violated merely because all processes have a few transitions that distinguish them from their peers.

We present an exact algorithm for reachability analysis, particularly efficient for *approximately fully symmetric* Kripke structures. The predicate whose reachability is being checked may be asymmetric, i.e., it may distinguish between components. Errors are discovered at minimum distance from the initial state, and concrete paths to them can be recovered, making the algorithm complete for safety property verification or falsification.

Organization. We begin by introducing the main ideas of this article by means of an example (section 2). We continue with mathematical preliminaries (section 3) and the computational model used (section 4). We then present the technical ingredients of the proposed technique (sections 5, 6). The algorithm itself is given in section 7, followed by the proof of its correctness (section 8), details on its implementation and efficiency (section 9), and an experimental evaluation (section 10). We conclude with a detailed discussion of related work (section 11) and then summarize this article in section 12.

2. An Example

Consider the variant of the *Readers-Writers problem* shown in figure 1. We assume 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. The first is executable by any process provided 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 ($s_3 \neq C$). The intuition is that readers, who only read, may enter their critical sections simultaneously, as long as the writer is not residing in its own critical section.

With each process initially in local state N, the induced (asymmetric) 3-process Kripke structure has 22 reachable states. We propose a method that, in contrast, 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

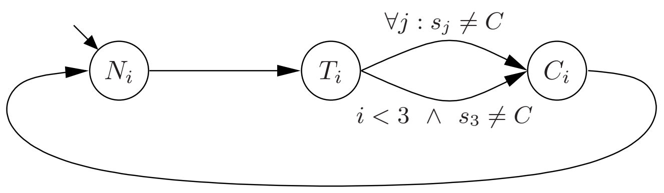


Fig. 1. Local state transition diagram of process i for an asymmetric system

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

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

local state tuple (X, Y, Z). Consider, for example, the abstract state *NNT*, representing (N, N, T), (N, T, N) and (T, N, N). The 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.

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 local transitions 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 for an index i 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) (both of which happen to be unreachable).

To keep the representation legible, the effect of subsumption is not shown in the reachability tree in figure 2, with one exception for illustration: consider again the abstract state *NNT*. The edge guarded $i < 3 \wedge s_3 \neq C$ is applicable to this state both for $i = 1$ and concrete state (T, N, N), as well as $i = 2$ and concrete state (N, T, N), respectively. The two resulting concrete successor states can be represented abstractly as $NC \mid N$. This abstract state is new, but is fortunately subsumed by the abstract state *NNC*, which is implicitly annotated by the set of all permutations on $\{1, 2, 3\}$. Therefore, abstract state $NC \mid N$ can be discarded. This example illustrates the scenario mentioned earlier: concrete state (N, C, N) is reachable along different paths, one that uses a symmetry-violating edge, and one that does not. We can ignore the former.

We finally remark that the structure induced by figure 1 is not *virtually* symmetric and hence not *nearly* or *roughly* so [EHT00, ET99, see section 11]. To see this, consider the (valid) transition $(T, C, T) \to (C, C, T)$. Applying transposition (2 3) to it we obtain the transition $(T, T, C) \to (C, T, C)$, which is invalid, but belongs to the structure's *symmetrization* [EHT00]. Virtual symmetry requires a way to permute the target state that makes the transition valid, which is impossible in this example. As a corollary, this structure is not bisimilar to its standard symmetry quotient, obtained by existential abstraction with respect to symmetry equivalence of states. Moreover, the reachable part of this quotient contains many states that are concretely unreachable, such as the state (N, C, C). The standard quotient is thus unsuitable even for reachability analysis: it strictly overapproximates the set of reachable states.

3. 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 $\{1, \ldots, n\}$, and let $\pi \in \mathit{Sym}_n$ act on a state $s \in S$ in the form $\pi(s_1, \ldots, s_n) = (s_{\pi(1)}, \ldots, s_{\pi(n)})$. We extend π to act on a transition $(s, t) \in R$ pointwise, in the form

$\pi((s,t)) = (\pi(s), \pi(t))$. M is said to be *symmetric* if for every $\pi \in \mathit{Sym}_n$, $\pi(R) = R$, that is,

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

A symmetric structure can be reduced to a smaller, bisimilar quotient structure based on the *orbit relation*: $s \equiv t$ iff $\exists \pi : \pi(s) = t$. Model-checking this quotient yields an exact and efficient verification procedure [CEFJ96, ES96].

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

We say a partition $\mathbb{P}$ *generates* a permutation π on $\{1, \ldots, n\}$ if for all $i \in \{1, \ldots, n\}$, $i \equiv_{\mathbb{P}} \pi(i)$. In section 4.2, we use the permutations generated by a partition to represent a subgroup of the automorphism group of a process transition. We therefore establish:

Property 1. The permutations generated by a partition $\mathbb{P}$ form a group, denoted $\langle \mathbb{P} \rangle$.

Proof: by a standard argument, which we include here for later reference.

Any partition generates the identity permutation. For closure under the group operation function composition (which is an associative operation), assume α, β are permutations generated by a given partition $\mathbb{P}$, and consider $i \in \{1, \ldots, n\}$. Since $\beta \in \langle \mathbb{P} \rangle$, i and $\beta(i)$ belong to the same cell in $\mathbb{P}$. Since $\alpha \in \langle \mathbb{P} \rangle$, $\beta(i)$ and $\alpha(\beta(i))$ belong to the same cell in $\mathbb{P}$. By transitivity, i and $\alpha(\beta(i))$ belong to the same cell in $\mathbb{P}$.
For closure under inversion, assume $\alpha \in \langle \mathbb{P} \rangle$; we show that the inverse permutation $\alpha^- : \{1, \ldots, n\} \to \{1, \ldots, n\}$ also belongs to $\langle \mathbb{P} \rangle$. To this end, let $P \in \mathbb{P}$ such that $i \in P$; we show $\alpha^-(i) \in P$. Consider the infinite sequence $\Omega = i, \alpha(i), \alpha^2(i), \ldots$ over $\{1, \ldots, n\}$. By induction over the exponent of α, all elements of this sequence belong to P. Since $\{1, \ldots, n\}$ is finite, the sequence contains a repetition, say $\alpha^r(i) = \alpha^s(i)$ for some r, s with $r < s$. Applying the function $(\alpha^{r+1})^-$ to this equality yields $\alpha^-(i) = \alpha^{s-r-1}(i)$. Since $s - r - 1 \geq 0$, the element $\alpha^{s-r-1}(i)$ belongs to Ω, so $\alpha^-(i) \in P$. ☐

In fact, a permutation group generated by a partition is a product of orthogonal subgroups of Sym_n. For example, the partition $|\,1,4\,|\,2,5\,|\,3,6\,|$ generates a group of order $2 \times 2 \times 2 = 8$. The coarsest partition $|\,1,\ldots,n\,|$ generates the entire symmetry group Sym_n. The finest partition $|\,1\,|\ldots|\,n\,|$ generates only the identity permutation.

4. Modeling Approximately Symmetric Systems

Our work targets concurrent systems with an asynchronous execution semantics; we refer to the concurrent components as *processes*. In this section we describe the system model, which allows asymmetry in the behavior of the processes. If the processes have very little behavior in common, the description can be as large as linear in their number.

4.1. Modeling Asymmetry

A system is asymmetric if it is not symmetric. By equation (1), this means that the condition $\pi(R) = R$ is not satisfied for all permutations. When we use the intuitive term *approximate symmetry*, we mean that for most transitions $r \in R$ and most permutations π, the condition $\pi(r) \in R$ holds.

Consider $r = (s,t) \in R$ and π such that $\pi(r) \notin R$. Suppose transition $r \in R$ is due to process i making a local transition, namely from s_i to t_i. The condition $\pi(r) \notin R$ then states that process $\pi(i)$ may not move from $s_{\pi(s)}$ to $t_{\pi(r)}$, given the local states of the other processes in s. Thus, to model symmetry violations in executing a transition, we need to consider the process identity and the global state context in which the transition is executed. We now introduce a computational model for describing approximately symmetric systems.

4.2. Computational Model

We assume we are given a parameterized local state transition diagram describing the behavior of a single process.[1] The overall system is understood as the concurrent execution of a number of instances of this template. Formally, 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

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

ϕ is a two-place predicate taking an index i and a state s. The state defines the context in which the edge is to be executed. The intended semantics is that $\phi(i,s)$ evaluates to *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. We say that ϕ is *symmetric* if, for any permutation π, $\phi(i,s) \Leftrightarrow \phi(\pi(i),\pi(s))$ is valid, and asymmetric otherwise. An edge labeled with $\phi,\mathbb{L}$ is said to be symmetric if ϕ is. For example, in figure 1 we have seen the predicate

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

It is asymmetric because for $i := 1$, $s := (N,T,C)$, and $\pi := (1\ 3)$ (which transposes indices 1 and 3), $\phi(i,s) \neq \phi(\pi(i),\pi(s))$. 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 $\pi = (1\ 2)$, i.e. $\phi(i,s) = \phi(\pi(i),\pi(s))$ for all i, s. In common variants of the r-readers/$(n-r)$-writers problem, the asymmetric edges are immune to any products of permutations of $\{1,\ldots,r\}$ and $\{r{+}1,\ldots,n\}$. Such permutations are generated by the partition $|\,1..r\,|(r{+}1)..n\,|$.

Symbol $\mathbb{L}$ 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 $\mathbb{L} = |\,1,2\,|\,3\,|$. In approximately symmetric systems, $\mathbb{L}$ is—for most edges—the coarsest partition, generating Sym_n. For the remaining edges—those that violate symmetry—we expect the user to provide an as-coarse-as-possible $\mathbb{L}$. The program text giving rise to the edge may suggest a group of automorphisms; see section 10 for an example. If deemed necessary, a propositional SAT-solver can aid the formal derivation 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 $M = (S,R)$: define $S := \{1,\ldots,l\}^n$ and R as the set of transitions $(s_1,\ldots,s_n) \to (t_1,\ldots,t_n)$ with the property that there is an index $i \in \{1,\ldots,n\}$ such that

1. there exist ϕ and $\mathbb{L}$ such that $s_i \xrightarrow{\phi,\mathbb{L}} t_i$ is an edge and $(i,(s_1,\ldots,s_n)) \models \phi$, and
2. $s_j = t_j$ for all j with $j \neq i$. (4)

We do not explicitly define a labeling function. Instead, atomic properties are formed over predicates specifying the local state of each process; explicit atomic propositions can be viewed as an abstraction of such predicates. Also note that $\mathbb{L}$ is vacuous in the definition of M; it is merely a user-supplied annotation that guides the exploration algorithm.

4.3. Shared Variables

In practice, system models usually rely on *shared variables* for communication. These may appear in the edge predicate ϕ, and they may be assigned as a result of a transition. Most shared variables have no bearing on the symmetry of the model. This includes synchronization variables such as semaphores, and the *busy* variable in the resource controller example used in section 10, which indicates whether a server is currently serving a request.

Some shared variables refer to process indices, such as one that stores the next client to be served in a client-server model. Conditions on, and assignments to, these *id-sensitive* variables may impact symmetry [EW03] and must thus be taken into account when determining a suitable partition $\mathbb{L}$ for an edge $A \xrightarrow{\phi,\mathbb{L}} B$.

[1] In practice, the local state transition diagram is of course not constructed explicitly. Instead, the reachable state space is generated on the fly from the high-level program.

Suppose after each service to a client, control must be passed back to the server, indicated by a pointer p reset to index 1 after each client transition. This pointer is an id-sensitive shared variable; the assignment $p := 1$ implies a partition label $\mathbb{L} := |\,1\,|\,2,\ldots,n\,|$ for the edge.

5. Annotating States with Partitions

The goal of this paper is an efficient exploration algorithm for the Kripke structure defined in the previous section. The formal search space of the exploration is the set $\widehat{S} := \{1,\ldots,l\}^n \times Part_n$, where $Part_n$ is the set of all partitions of $\{1,\ldots,n\}$. The algorithm accumulates states annotated with partitions that indicate how symmetry was violated in reaching this state. 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 precisely those generated by the partition. This naturally gives rise to the following definition of a permutation action on an annotated state:

Definition 2. Let π be a permutation on $\{1,\ldots,n\}$. For a state $s = (s_1,\ldots,s_n) \in \{1,\ldots,l\}^n$, let π act on s as defined in section 3. The *extension of* π to act on an element $\hat{s} = (s,\mathbb{P})$ of $\widehat{S}$ is defined as

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

Note that a permutation never changes the partition a state is annotated with. Extending the orbit relation to $\widehat{S}$, we write $\hat{s} \equiv \hat{t}$ if there exists π such that $\pi(\hat{s}) = \hat{t}$. The notions of *permutation action* and *orbit relation* thus have distinct meanings in S and $\widehat{S}$; if necessary, the domain will be explicitly qualified in the text.

It turns out that the permutation action from definition 2 is not a *group action*: there exist permutations π and σ and a state $\hat{s} \in \widehat{S}$ such that $\sigma(\pi(\hat{s})) \neq (\sigma \circ \pi)(\hat{s})$.[2] The group action property would imply that the permutation action is a bijection, and that the orbit relation is an equivalence relation on $\widehat{S}$. Without being an equivalence, the orbit relation would not even give rise to a meaningful existential abstraction. Fortunately, even without the group action property, we can still show:

Property 3. The permutation action from definition 2 is a bijection on $\widehat{S}$. Further, the orbit relation on $\widehat{S}$ is an equivalence on $\widehat{S}$.

Proof:

(1) Bijectivity of $\pi\colon \widehat{S} \to \widehat{S}$: For injectivity, first observe that $\pi(s,\mathbb{P}) = \pi(t,\mathbb{Q})$ implies $\mathbb{P} = \mathbb{Q}$; hence either both $\mathbb{P}$ and $\mathbb{Q}$ generate π, or neither does. Since π is a bijection on $S = \{1,\ldots,l\}^n$, $s = t$ follows in both cases of definition 2. For surjectivity, consider $(s,\mathbb{P})$. If $\pi \in \langle\mathbb{P}\rangle$, then $\pi(\pi^-(s),\mathbb{P}) = (\pi(\pi^-(s)),\mathbb{P}) = (s,\mathbb{P})$. Otherwise, $\pi(s,\mathbb{P}) = (s,\mathbb{P})$.

(2) We show that $\equiv$ is an equivalence relation on $\widehat{S}$:

reflexivity: The identity permutation id on $\{1,\ldots,n\}$ satisfies $id(\hat{s}) = \hat{s}$.

symmetry: Let $\pi(\hat{s}) = \hat{t}$, say $\hat{s} = (s,\mathbb{P})$, $\hat{t} = (t,\mathbb{P})$. Note that $\pi \in \langle\mathbb{P}\rangle$ iff $\pi^- \in \langle\mathbb{P}\rangle$ (see proof of property 1). We show $\pi^-(\hat{t}) = \hat{s}$:

(a) If $\pi \in \langle\mathbb{P}\rangle$, then $t = \pi(s)$, thus $\pi^-(t,\mathbb{P}) = (\pi^-(t),\mathbb{P}) = (s,\mathbb{P}) = \hat{s}$.

(b) Otherwise, $t = s$, hence $\pi^-(t,\mathbb{P}) = (t,\mathbb{P}) = (s,\mathbb{P}) = \hat{s}$.

transitivity: Let $\pi(\hat{s}) = \hat{t}$, $\sigma(\hat{t}) = \hat{u}$, which implies that all three states have the same annotation, say $\hat{s} = (s,\mathbb{P})$, $\hat{t} = (t,\mathbb{P})$, $\hat{u} = (u,\mathbb{P})$. We have to find a permutation ρ such that $\rho(\hat{s}) = \hat{u}$.

(a) If $\mathbb{P}$ generates both π and σ, then $\pi(s) = t$, $\sigma(t) = u$. Choose $\rho := \sigma \circ \pi$, and we obtain $\rho(\hat{s}) = \sigma(\pi(s),\mathbb{P}) = (\sigma(t),\mathbb{P}) = (u,\mathbb{P}) = \hat{u}$.

[2] Consider the index set $\{1,2,3\}$, let π be the left-shift permutation, let σ transpose 1 and 2, and let $\hat{s} = ((A,B,C),|\,1,2\,|\,3\,|)$.

(b) If $\pi \in \langle \mathbb{P} \rangle$, but $\sigma \notin \langle \mathbb{P} \rangle$, then $\pi(s) = t = u$. Choose $\rho := \pi$, and we obtain $\rho(\hat{s}) = \pi(\hat{s}) = (\pi(s), \mathbb{P}) = (u, \mathbb{P}) = \hat{u}$.

(c) If $\sigma \in \langle \mathbb{P} \rangle$, but $\pi \notin \langle \mathbb{P} \rangle$, then $s = t$, $\sigma(t) = u$. Choose $\rho := \sigma$, and we obtain $\rho(\hat{s}) = \sigma(\hat{s}) = (\sigma(s), \mathbb{P}) = (\sigma(t), \mathbb{P}) = (u, \mathbb{P}) = \hat{u}$.

(d) If $\mathbb{P}$ generates neither π nor σ, then $s = t = u$. Choose $\rho := id$, and we obtain $\rho(\hat{s}) = \hat{s} = \hat{u}$. □

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 $\widehat{S}$ and in S:

Definition 4. The *orbit* of a state $\hat{s} = (s, \mathbb{P}) \in \widehat{S}$ is defined as

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

If $t \in Orbit(\hat{s})$, we say $\hat{s}$ *represents* t.

If $\mathbb{P}$ is the coarsest partition $|\,1, \ldots, n\,|$, then $Orbit(s, \mathbb{P})$ reduces to the equivalence class that s belongs to under the standard orbit relation on S. If $\mathbb{P}$ is the finest partition $|\,1\,|\ldots|\,n\,|$, then $Orbit(s, \mathbb{P}) = \{s\}$.

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

state	size of its orbit			
$(ABCD,\ \|1,2,3,4\|)$	$4!$	$=$	24	(standard symmetry)
$(ABCD,\ \|1,2\|3,4\|)$	$2! \times 2!$	$=$	4	
$(ABCD,\ \|1,2\|3\|4\|)$	$2! \times 1! \times 1!$	$=$	2	
$(ABCD,\ \|1\|2\|3\|4\|)$	$1! \times 1! \times 1! \times 1!$	$=$	1	

Canonization. Algorithms exploring state spaces under symmetry typically replace each encountered state by a unique representative of its orbit; this facilitates the detection of states identical up to permutations. In our case, the representative mapping must respect the partition boundaries in each abstract state. More precisely, given some total order on the set of local states, we call an abstract state $(s, \mathbb{P})$ *canonical* if for each cell $P \in \mathbb{P}$, the projection of s onto the indices in P forms a non-decreasing sequence of local states. For example, using the lexicographic ordering, the state $(ADBC, |\,1, 2\,|\,3, 4\,|)$ is in canonical form, whereas $(ADBC, |\,1, 2, 3\,|\,4\,|)$ is not. The latter state can, however, be *canonized* to $(ABDC, |\,1, 2, 3\,|\,4\,|)$. Two abstract states with the same canonization have the same orbits and thus represent the same set of concrete states. Canonization is therefore a suitable representative mapping.

Discussion. Annotations are frequently used to store additional information on a state [SG04, for example]. They incur, however, an increase of the *conceivable* state space. The information they provide must be used aggressively to not only compensate for this increase, but to in fact *decrease* the size of the *reachable* state space. In the case of the exploration algorithm we describe in section 7, the annotations inform about the history that led to the state. Our algorithm occasionally has to refine a state by refining its partition if it determines that future exploration differs between members of its orbit. Without any further mechanism, the sizes of the orbits and thus the compression rate of the algorithm would gradually decrease. The idea to counter this effect is that, although the refinement may be necessary at the current point of the exploration, the refined state may later turn out to be redundant. We describe this idea next.

6. Subsumption

Orbits in standard symmetry reduction are equivalence classes and, as such, are either disjoint or equal. In contrast, the new orbit definition is not based on an equivalence relation. Indeed, the orbits of the four states in the table below definition 4 form a strictly descending chain with respect to the subset ordering. If they are encountered during exploration and we only care about reachability of the represented concrete states, it is unnecessary to remember all four states: the first *subsumes* the other three.

Definition 5. State $\hat{s} \in \widehat{S}$ *subsumes* $\hat{t} \in \widehat{S}$, 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:
$(ABC,\ \lvert 1,2,3 \rvert)$	$(ABC,\ \mathbb{Q})$, $(BCA,\ \mathbb{Q})$	$(ABB,\ \mathbb{Q})$
$(ABC,\ \lvert 1,2 \lvert 3 \rvert)$	$(ABC,\ \lvert 1 \lvert 2 \lvert 3 \rvert)$, $(BAC,\ \lvert 1 \lvert 2 \lvert 3 \rvert)$	$(CBA,\ \mathbb{Q})$
$(ABC,\ \lvert 1 \lvert 2 \lvert 3 \rvert)$	itself only	$(ABC,\ \lvert 1,2 \lvert 3 \rvert)$

Definition 5 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 6. State $\hat{s} = (s, \mathbb{P})$ subsumes state $\hat{t} = (t, \mathbb{Q})$ exactly if

1. for all $i, j \in \{1, \ldots, n\}$, $(i \equiv_{\mathbb{Q}} j) \Rightarrow (i \equiv_{\mathbb{P}} j \vee t_i = t_j)$ is valid, and
2. $t \in \mathit{Orbit}(\hat{s})$.

Remark. The implication in condition 1 is slightly weaker than $(i \equiv_{\mathbb{Q}} j) \Rightarrow (i \equiv_{\mathbb{P}} j)$, which states that $\mathbb{P}$ is at least as coarse as $\mathbb{Q}$. To illustrate why the term $t_i = t_j$ is needed in 1. to achieve an equivalent characterization of subsumption, consider $\hat{s} = (AAB, \lvert 1 \lvert 2 \lvert 3 \rvert)$, which has a finer partition than $\hat{t} = (AAB, \lvert 1,2 \lvert 3 \rvert)$, but subsumes $\hat{t}$, according to definition 5.

Proof of theorem 6.

$\Rightarrow$: Assume $\mathit{Orbit}(\hat{s}) \supseteq \mathit{Orbit}(\hat{t})$. Condition 2 of the theorem follows from $t \in \mathit{Orbit}(\hat{t})$. Regarding condition 1, consider i, j with $i \equiv_{\mathbb{Q}} j$. If $t_i = t_j$, the implication is proved. Assume now $t_i \neq t_j$, and let $\pi = (i\ j)$, the transposition of i and j. This permutation is generated by $\mathbb{Q}$, thus $\pi(t) \in \mathit{Orbit}(\hat{t}) \subseteq \mathit{Orbit}(\hat{s})$. Therefore, there exists a permutation $\beta \in \langle \mathbb{P} \rangle$ that satisfies $\beta(s) = \pi(t)$. Since $t \in \mathit{Orbit}(\hat{s})$, let $\sigma \in \langle \mathbb{P} \rangle$ such that $\sigma(s) = t$. We get $\beta(\sigma^{-}(t)) = \beta(s) = \pi(t)$. Thus, we have found a permutation $\alpha := \beta \circ \sigma^{-}$ that satisfies $\alpha(t) = \pi(t)$ and $\alpha \in \langle \mathbb{P} \rangle$ since $\beta, \sigma \in \langle \mathbb{P} \rangle$.
Now consider the sequence $i, \alpha(i), \alpha^2(i), \ldots$. Since $\alpha \in \langle \mathbb{P} \rangle$, the sequence's elements belong to the same cell within $\mathbb{P}$. The goal is to show that j is part of this sequence, which shows that $i \equiv_{\mathbb{P}} j$ and thus completes the proof.
By a cancellation argument similar to the one used in the proof of property 1, there is an index $x > 0$ such that $i = \alpha^x(i)$. Now consider the local state sequence

$$t_i\ ,\quad t_{\alpha(i)}\ ,\quad t_{\alpha^2(i)}\ ,\quad \ldots\ ,\quad t_{\alpha^x(i)} = t_i\ .$$

This sequence begins and ends with the same local state t_i. Since $t_{\alpha(i)} = t_{\pi(i)} = t_j$, it also contains an element that differs from t_i, namely t_j. Thus, there is an index k such that $t_{\alpha^k(i)} \neq t_i$, and $t_{\alpha^{k+1}(i)} = t_i$. Local state $t_{\alpha^k(i)}$ is the $\alpha^k(i)$'th element of t, and $t_{\alpha^{k+1}(i)}$ is the $\alpha^k(i)$'th element of $\alpha(t)$. Recalling that t and $\alpha(t)$ are identical except for positions i and j, it follows that $\alpha^k(i) = i$ or $\alpha^k(i) = j$. Since $t_{\alpha^k(i)} \neq t_i$, we conclude $\alpha^k(i) \neq i$ and thus $\alpha^k(i) = j$.

$\Leftarrow$: Assume conditions 1 and 2, and consider $u \in \mathit{Orbit}(\hat{t})$; we show $u \in \mathit{Orbit}(\hat{s})$. Let $\pi \in \langle \mathbb{Q} \rangle$ be such that $\pi(t) = u$. According to 2., we have some $\sigma \in \mathbb{P}$ such that $\sigma(s) = t$. We have to find a permutation $\alpha \in \langle \mathbb{P} \rangle$ that satisfies $\alpha(s) = u$. A first choice is $\alpha := \pi \circ \sigma$, since then $\alpha(s) = \pi(\sigma(s)) = \pi(t) = u$. However, π may not be generated by $\mathbb{P}$, thus α may not be either. We construct a permutation π' with the following requirements:

Requirement 1: $\pi'(t) = \pi(t)$, i.e. for all i, $t_{\pi'(i)} = t_{\pi(i)}$, and

Requirement 2: $\pi' \in \langle \mathbb{P} \rangle$, i.e. for all i, $i \equiv_{\mathbb{P}} \pi'(i)$.

Once we have π' satisfying these requirements, choosing $\alpha := \pi' \circ \sigma$ completes the proof, since $\mathbb{P}$ generates both π' and σ, and $\alpha(s) = \pi'(\sigma(s)) = \pi'(t) = \pi(t) = u$.
To construct π', let $i \in \{1, \ldots, n\}$, and let $Q \in \mathbb{Q}$ be the cell containing i. We distinguish two cases:

(a) Q is fully contained in some $P \in \mathbb{P}$. In this case, define $\pi'(i) = \pi(i)$. Then $t_{\pi'(i)} = t_{\pi(i)}$ and $i \equiv_{\mathbb{P}} \pi'(i)$ (since $\pi \in \langle \mathbb{Q} \rangle$, $i, \pi(i) \in Q \subseteq P$).

(b) Q is not fully contained in any cell of $\mathbb{P}$. We show that in this case $t_i = t_j$ for all $j \in Q$. We have

$i \equiv_{\mathbb{Q}} j$. If $i \not\equiv_{\mathbb{P}} j$, then $t_i = t_j$ by condition 1. If $i \equiv_{\mathbb{P}} j$, then let $k \in Q$ such that $j \not\equiv_{\mathbb{P}} k$. Such a k exists since Q is not contained in any cell of $\mathbb{P}$. We have $i \equiv_{\mathbb{Q}} j \equiv_{\mathbb{Q}} k$, hence $t_i = t_k$ and $t_j = t_k$ by separate applications of condition 1, thus $t_i = t_j$.
We define $\pi'(i) = i$. Then $t_{\pi'(i)} = t_i = t_{\pi(i)}$ (since π only moves elements within Q, which are all equal) and $i \equiv_{\mathbb{P}} \pi'(i) = i$.

Finally, π' is a permutation on $\{1, \ldots, n\}$ since it is identical to π on all cells from $\mathbb{Q}$ fully contained in some cell in $\mathbb{P}$, and is the identity on all other cells. □

Condition 1 of theorem 6 can be verified in $\mathcal{O}(n^2)$ worst-case time, using partition data structures that allow deciding $i \equiv_{\mathbb{P}} j$ in constant time. In practice, violations of condition 1 are often detected much faster using heuristics such as comparing the numbers of cells in $\mathbb{P}$ 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 verify that the projections are the same up to permutation, using counting sort.

Algebraic properties of subsumption. The relation $\triangleright$ is a *preorder*: it is reflexive and transitive. It is, however, neither symmetric (e.g. $(AB, |\,1,2\,|) \triangleright (AB, |\,1|2\,|)$ but not vice versa) nor anti-symmetric (e.g. $(AB, |\,1,2\,|)$ and $(BA, |\,1,2\,|)$ subsume each other but differ). Thus, it is neither an equivalence nor a partial order. We can, however, derive an equivalence relation from $\triangleright$ by making it bidirectional: two states $\hat{s}, \hat{t} \in \widehat{S}$ are *subsumption equivalent*, denoted $\hat{s} \bowtie \hat{t}$, if $\hat{s} \triangleright \hat{t}$ and $\hat{t} \triangleright \hat{s}$, that is, if their orbits are equal. The following lemma elucidates the relationship between the subsumption equivalence and the orbit equivalence on $\widehat{S}$ from definition 2.

Lemma 7. For any $\hat{s}, \hat{t} \in \widehat{S}$, $\hat{s} \equiv \hat{t}$ implies $\hat{s} \bowtie \hat{t}$.

Proof:
Let as usual $\hat{s} = (s, \mathbb{P})$, $\hat{t} = (t, \mathbb{Q})$. Suppose $\pi(\hat{s}) = \hat{t}$ for some π. We show that $\hat{s} \triangleright \hat{t}$; the result $\hat{t} \triangleright \hat{s}$ follows with a symmetric argument.
If $\pi \in \langle \mathbb{P} \rangle$, then $\pi(\hat{s}) = (\pi(s), \mathbb{P}) = (t, \mathbb{Q})$. Thus $\mathbb{P} = \mathbb{Q}$, which establishes condition 1 of theorem 6, and $\pi(s) = t$, which establishes condition 2. If $\pi \notin \langle \mathbb{P} \rangle$, then $\pi(\hat{s}) = (s, \mathbb{P}) = (t, \mathbb{Q})$, so $\hat{s} = \hat{t}$, which establishes $\hat{s} \triangleright \hat{t}$ by reflexivity of $\triangleright$. □

The converse of the lemma does not hold; consider the states $(AAB, |\,1|2|3\,|)$ and $(AAB, |\,1,2|3\,|)$ (see also the remark below theorem 6). Thus, the subsumption equivalence $\bowtie$ is coarser than $\equiv$, it relates more states. Moreover, the algorithm we present in the next section does not even require mutual subsumption: if $\hat{s}$ subsumes $\hat{t}$, the latter will be discarded. Subsumption in one direction achieves strictly more compression than the orbit relation. To summarize, in general we have

$$\equiv \;\subsetneq\; \bowtie \;\subsetneq\; \triangleright\,.$$

In perfectly symmetric systems, where each state is (implicitly) annotated with the coarsest partition $|\,1, \ldots, n\,|$, the three relations coincide.

Discussion. In this and the previous section, we introduced the notions of state annotation by partitions, and subsumption. Roughly speaking, the annotations ensure the exactness of the exploration algorithm proposed in the next section, as they prevent the abstraction from becoming an overapproximation. Subsumption, on the other hand, contributes to efficiency, as it captures redundancy in the exploration that was not foreseeable at the time an abstract state was created, and uses it to prune the search. It is the symbiosis of the two notions that distinguishes our exploration algorithm, which we present next.

7. Lazy Symmetry Reduction

We are now ready to present our algorithm for state space exploration on the (approximately symmetric) structure $M = (S, R)$. The goal is to compute the set of states of M that are reachable from some initial state $\lambda \in S$. Algorithm 1 below actually accumulates elements of $\widehat{S}$, i.e. states annotated with partitions that indicate how symmetry was violated in reaching this state. In section 8 we formalize the relationship between the states reachable in M and the states found by the algorithm.

Algorithm 1 Lazy Symmetry Reduction

```
Input:  initial state $\lambda \in S$
1: $\mathit{Initialize}(\mathbb{P}_0), \mathit{Unexplored} :=$ queue $(\lambda, \mathbb{P}_0)$, $\mathit{Reached} :=$ list $(\lambda, \mathbb{P}_0)$
2: while $\mathit{Unexplored} \neq \emptyset$ do
3:     dequeue a state $\hat{s} = (s, \mathbb{P})$ from $\mathit{Unexplored}$
4:     for all edges $e = A \xrightarrow{\phi,\mathbb{L}} B$ do
5:         $\mathbb{Q} := \mathit{glb}(\mathbb{P}, \mathbb{L})$
6:         $U := \mathit{unwind}(s, \mathbb{P}, \mathbb{L})$
7:         for all states $u \in U$ do
8:             for all cells $Q \in \mathbb{Q}$ do
9:                 if $\exists i \in Q : u_i = A \ \wedge\ (i, u) \models \phi$ then
10:                    $v := (u_1, \ldots, u_{i-1}, B, u_{i+1}, \ldots, u_n)$
11:                    $(t, \mathbb{Q}) := \mathit{canonize}(v, \mathbb{Q})$
12:                    $\mathit{update}(t, \mathbb{Q})$

Subroutine $\mathit{update}(t, \mathbb{Q})$: updating $\mathit{Reached}$ and $\mathit{Unexplored}$
Input: newly computed state $\hat{t} = (t, \mathbb{Q})$
i:   if no state in $\mathit{Reached}$ subsumes $\hat{t}$
ii:      if $\hat{t}$ represents a concrete error state
iii:         return concrete error path and exit
iv:      remove from $\mathit{Unexplored}$ each state $\hat{w}$ that $\hat{t}$ subsumes: $\hat{t} \rhd \hat{w}$
v:       add $\hat{t}$ to $\mathit{Reached}$; enqueue $\hat{t}$ into $\mathit{Unexplored}$
```

Annotating the initial state. In line 1, the concrete initial state λ is annotated with a partition $\mathbb{P}_0$ that reflects the symmetry λ exhibits. $\mathbb{P}_0$ consists of one cell for each local state that appears in λ; the cell contains all indices of processes residing in this local state. For instance, for $\lambda = (A, C, A, B)$, we let $\mathbb{P}_0 = |\,1, 3\,|\,2\,|\,4\,|$. In practice, the initial state is often fully symmetric, i.e. of the form $\lambda = (A, \ldots, A)$. In this case, $\mathbb{P}_0$ reduces to the coarsest partition $|\,1, \ldots, n\,|$. Of course, the actual symmetry group of M depends on much more than the initial state — the algorithm *lazily* postpones worrying about the precision of the reduction.

A finer initial partition may be appropriate if the symmetry group of the system at hand is (approximately) a product of disjoint fully symmetric subgroups of Sym_n. Suppose the system is orthogonally composed of n_1 processes of type 1 and n_2 processes of type 2, and the processes across the types share no behavior other than the common initial local state A. In this case we can choose $\mathbb{P}_0 = |\,1, \ldots, n_1\,|\,n_1+1, \ldots, n_1+n_2\,|$ as the initial partition. This prevents algorithm 1 from wasting time by considering permutations across the two disjoint index sets. In practice, however, it is generally more useful to choose a generous value for $\mathbb{P}_0$ and refine it later as needed, especially when there is at least some common behavior across the process types, as is case in the examples in figure 1 and section 10. Note that in any case, the orbit of the abstract initial state $(\lambda, \mathbb{P}_0)$ is a singleton.

The abstract initial state is put on the *Unexplored* queue and the *Reached* list. While available, one state $\hat{s}$ is selected from *Unexplored* for expansion (lines 2, 3). Successors of $\hat{s} = (s, \mathbb{P})$ 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 $\mathbb{L}$, expressing violations about to be caused by edge e. Function *glb* in line 5 determines the partition $\mathbb{Q}$ such that $\langle\mathbb{Q}\rangle = \langle\mathbb{P}\rangle \cap \langle\mathbb{L}\rangle$. Partition $\mathbb{Q}$ is the greatest lower bound (meet) of $\mathbb{P}$ and $\mathbb{L}$ in the complete partition lattice, which uses "at-most-as-coarse-as" as the partial order relation. Again, the algorithm behaves lazily, in that the two incompatible partition annotations $\mathbb{P}$ and $\mathbb{L}$ are reconciled to a minimal extent: $\mathbb{Q}$ is the coarsest, "most optimistic" partition that is not coarser than $\mathbb{P}$ and $\mathbb{L}$.

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

$$\bigcup_{u \in U} \mathit{Orbit}(u, \mathbb{Q}) = \mathit{Orbit}(s, \mathbb{P}) . \tag{5}$$

It is always possible to find such a set U; a trivial choice is $U = \{\pi(s) : \pi \in \langle \mathbb{P} \rangle\} = \mathit{Orbit}(s, \mathbb{P})$. The objective is of course to find a *small* set U with property (5). Function *unwind* in line 6 returns the set $U = \{s\} \cup \{\pi(s) : \pi \in \langle \mathbb{P} \rangle \setminus \langle \mathbb{L} \rangle\}$, which is shown in section 8 to satisfy (5). Line 6 is the price we pay for the laziness. Fortunately, the *unwind* procedure is only a worst-case scenario; we discuss in section 9 how to avoid it in most cases, and how to perform it as efficiently as possible in the remaining cases.

Processes with indices in different cells of $\mathbb{Q}$ 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 (i, u) satisfies ϕ. If so, we let the process proceed, resulting in a new state v (line 10). As pointed out in section 5, it is advisable to keep each abstract state canonical, as is common in symmetry reduction. In line 11, $(t, \mathbb{Q})$ is obtained as the canonization of $(v, \mathbb{Q})$; this only requires finding the correct lexicographic position of the new element B, and only among the positions in Q.

The call to *update* in line 12 determines whether to add a new state $\hat{t} = (t, \mathbb{Q})$ to *Unexplored* and *Reached*, as defined in the second part of algorithm 1. If some state in *Reached* subsumes $\hat{t}$, nothing needs to be done; this also covers the case that $\hat{t}$ itself already belongs to *Reached*. Otherwise, $\hat{t}$ is checked for errors (line ii, discussed below). If none are found, states that $\hat{t}$ subsumes are removed from *Unexplored*: such states are implicitly explored as part of $\hat{t}$ and are thus redundant.[3] Finally, $\hat{t}$ is added to both containers.

Error Checking

The error condition to be checked in line ii of subroutine *update* need not be symmetric; it can be an arbitrary formula $E(t)$ in some efficiently decidable logic. Often E is simply a property over a single index. For example, suppose it is an error for process 3 to ever enter local state X: $E(t) = (t_3 = X)$. Line ii determines the unique cell $Q \in \mathbb{Q}$ such that $3 \in Q$. An error is reported exactly if the property $\exists p \in Q : t_p = X$ evaluates to *true*.

For multi-index properties, say over indices i and j, we determine the unique cells Q_i and Q_j that contain i and j, respectively, and then check the formula $\exists p \in Q_i,\, q \in Q_j : E$, where occurrences of the constants i and j in E as process indices are replaced by p and q, respectively.

If M has an error at distance d from λ, then algorithm 1 finds an abstract path of length d to a state that represents the concrete error state, as we prove in the next section. In particular, shortest counterexample paths are preserved. In line iii, a concrete error path is obtained by reversing the abstract transitions from $\hat{t}$ back to the abstract initial state, and lifting each abstract transition as follows. In each step, we pick a concrete state s in the current abstract state's orbit that has a concrete transition to the previously picked concrete state. We prove in section 8.2 that such a concrete predecessor s can always be found; see the comments at the end of that subsection.

Remark. Regarding line iv of algorithm 1, the only reason not to remove $\hat{w}$ from *Reached* (only from *Unexplored*) is to retain the ability to trace encountered errors back to the initial state, for which previously subsumed states may be needed. They are not needed for merely deciding the presence or absence of errors.

8. Correctness

In this section we show the efficacy of algorithm 1 for reachability analysis, by establishing a relationship between the states reachable in the concrete structure M as defined in equation (4), and the abstract states collected by the algorithm. We split the proof into four steps:

(section 8.1) Define an abstract transition system $\widehat{M}$ partially explored by algorithm 1.

(section 8.2) Show *finite-path correspondence* between M and $\widehat{M}$.

[3] *Unexplored* is thus a queue with a special "erase-from-the-middle" method.

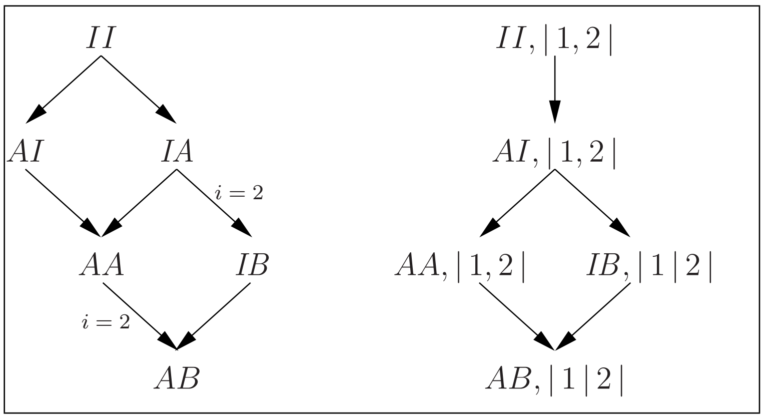


Fig. 3. Kripke structure and abstract structure induced by the skeleton $I \longrightarrow A \overset{i=2,\mid 1\mid 2\mid}{\longrightarrow} B$

(section 8.3) Show that algorithm 1 *without* subsumption implements reachability analysis on $\widehat{M}$.
(section 8.4) Show that algorithm 1 (i.e. *with* subsumption) implements reachability analysis on M.

In this section we denote the length of a finite path p (number of its edges) by $|p|$ and the ith state along the path by p^i. In particular, p^0 and $p^{|p|}$ denote the first and the last state of p, respectively. We say a state $t \in S$ is *reachable in M with depth d* if t is reachable and d is the length of a *shortest* path in M to t. We are interested in reachability from the initial state λ of M. Thus, when we speak of a path p in M, we mean a path that starts from λ.

8.1. Transition System $\widehat{M}$

We define a transition system that is partially explored by algorithm 1. To simplify the notation, we write "$(s,t) \in R$ via $\mathbb{L}$" to mean that the transition (s,t) is due to an edge in the local state transition diagram that is labeled with partition $\mathbb{L}$.

Definition 8. Let $\widehat{M} = (\widehat{S}, \widehat{R})$ be the abstract transition system defined by $\widehat{S} = S \times \mathit{Part}_n$ and

$$\begin{aligned}\widehat{R} \;=\; \{(\hat{s} = (s,\mathbb{P}),\ \hat{t} = (t,\mathbb{Q})) :\quad & \hat{s} \text{ and } \hat{t} \text{ are canonical, and there exist } \tilde{s} \in \mathit{Orbit}(\hat{s}),\ \tilde{t} \in \mathit{Orbit}(\hat{t}),\ \mathbb{L}\text{:}\\ & (\tilde{s},\tilde{t}) \in R \text{ via } \mathbb{L} \text{ and } \mathbb{Q} = \mathit{glb}(\mathbb{P},\mathbb{L})\}\,.\end{aligned}$$

Figure 3 shows on the left the reachable part of the 2-process Kripke structure M induced by the local state transition diagram $I \longrightarrow A \overset{i=2,\mid 1\mid 2\mid}{\longrightarrow} B$ (initial state I), in which the local transition from I to A is unconstrained, while only the second process ($i = 2$) is allowed to move from local state A to B. Solely for informational purposes, transitions in M that are due to the restricted edge are labeled in the figure with "$i = 2$". On the right is the abstract transition system $\widehat{M}$ according to definition 8. All abstract states are canonical. As an illustration, the abstract transition from $(AI, \mid 1,2 \mid)$ to $(IB, \mid 1 \mid 2 \mid)$ is justified by definition 8 with $\tilde{s} := \mathit{IA} \in \mathit{Orbit}(AI, \mid 1,2 \mid)$, $\tilde{t} := \mathit{IB} \in \mathit{Orbit}(IB, \mid 1 \mid 2 \mid)$ and $\mathbb{L} := \mid 1 \mid 2 \mid$. Indeed, $(\tilde{s},\tilde{t}) \in R$ via $\mathbb{L}$ and $\mathbb{Q} = \mid 1 \mid 2 \mid = \mathit{glb}(\mid 1,2 \mid, \mid 1 \mid 2 \mid) = \mathit{glb}(\mathbb{P},\mathbb{Q})$.

Remark. Kripke structure $\widehat{M}$ is not equipped with a labeling function, for the following reason. Under genuine symmetry, atomic propositions are required to be invariant under all applicable permutations, as is the case for the fully symmetric proposition $\exists i,j : i \neq j \wedge C_i \wedge C_j$. In our lazy approach, the set of applicable permutations shrinks from state to state as the exploration progresses. We therefore stipulate neither a logic fragment for expressing atomic propositions, nor a labeling function. This omission is, however, without consequences, as algorithm 1 only claims to explore reachable states, rather than perform full-fledged temporal-logic model checking.

8.2. Finite-Path Correspondence Between M and $\widehat{M}$

The relationship between M and $\widehat{M}$ is not as strong as one typically enjoys with symmetry reduction: the structures are not bisimilar; we discuss this at the end of section 8.2. This is in part because it is not possible to assign atomic propositions to abstract states in a meaningful way. Instead, we strive for a purely graph-based relationship, which we call *finite-path correspondence*. This correspondence holds in both directions as follows.

Lemma 9. For every finite path p in M, there is a path $\hat{p}$ in $\widehat{M}$ of the same length such that for all $i \leq |p|$, $p^i \in \mathit{Orbit}(\hat{p}^i)$.

Proof by induction on the length m of p.

$m = 0$: We have $p^0 = \lambda \in \mathit{Orbit}(\lambda, \mathbb{P}_0) = \hat{p}^0$, where $\hat{p}$ is the only path in $\widehat{M}$ of length 0, namely $(\hat{s})$ with $\hat{s} = (\lambda, \mathbb{P}_0)$.

$m \to m+1$: Suppose the statement holds for all paths in M of length m, and consider path p of length $m+1$. Let $s := p^m$ and $t := p^{m+1}$, and suppose $(s,t) \in R$ via $\mathbb{L}$. Consider the prefix q of p up to p^m. By the induction hypothesis, there is path $\hat{q}$ in $\widehat{M}$ of length m leading to an element $\hat{q}^m = (v, \mathbb{P})$, and $s \in \mathit{Orbit}(v, \mathbb{P})$. Let $\mathbb{Q} = \mathit{glb}(\mathbb{P}, \mathbb{L})$ and $\hat{t} = \mathit{canonize}(t, \mathbb{Q})$. Then there is $\tilde{s} := s \in \mathit{Orbit}(v, \mathbb{P})$, $\tilde{t} := t \in \mathit{Orbit}(t, \mathbb{Q}) = \mathit{Orbit}(\hat{t})$ such that $(\tilde{s}, \tilde{t}) = (s,t) \in R$ via $\mathbb{L}$ and $\mathbb{Q} = \mathit{glb}(\mathbb{P}, \mathbb{L})$. Thus $((v, \mathbb{P}), \hat{t}) \in \widehat{R}$ by definition 8, and we can build $\hat{p}$ of length $m+1$ by appending $\hat{t}$ to $\hat{q}$. $\square$

Lemma 10. For every finite path $\hat{p}$ in $\widehat{M}$ and any $u \in \mathit{Orbit}(\hat{p}^{|\hat{p}|})$, there is a path p in M of the same length such that for all $i \leq |p|$, $p^i \in \mathit{Orbit}(\hat{p}^i)$ and $p^{|p|} = u$.

Recall that $\hat{p}^{|\hat{p}|}$ and $p^{|p|}$ are the final states of finite paths $\hat{p}$ and p, respectively.

Proof by induction on the length m of $\hat{p}$.

$m = 0$: Then $\hat{p}$ consists solely of $\hat{\lambda} = (\lambda, \mathbb{P}_0)$, and $u \in \mathit{Orbit}(\hat{p}^0) = \hat{\lambda}$ implies $u = \lambda$, since the orbit of $\hat{\lambda}$ is a singleton. We let p consist solely of λ. The condition $p^0 \in \mathit{Orbit}(\hat{p}^0)$ is satisfied by construction.

$m \to m+1$: Suppose $\hat{p}$ has length $m+1$. Let $\hat{p}^m = (s, \mathbb{P})$ and $\hat{p}^{m+1} = (t, \mathbb{Q})$. By construction, there are $\tilde{s} \in \mathit{Orbit}(s, \mathbb{P})$ and $\tilde{t} \in \mathit{Orbit}(t, \mathbb{Q})$ such that $(\tilde{s}, \tilde{t}) \in R$ via some $\mathbb{L}$ and $\mathbb{Q} = \mathit{glb}(\mathbb{P}, \mathbb{L})$. Further, let $u \in \mathit{Orbit}(t, \mathbb{Q})$ be arbitrary, say $u = \pi(t)$ for $\pi \in \langle \mathbb{Q} \rangle$. Also $t = \sigma(\tilde{t})$ for some $\sigma \in \langle \mathbb{Q} \rangle$.
Since $\langle \mathbb{Q} \rangle \subseteq \langle \mathbb{L} \rangle$ and thus $\sigma \in \langle \mathbb{L} \rangle$, we conclude from $(\tilde{s}, \tilde{t}) \in R$ via $\mathbb{L}$ that $\sigma(\tilde{s}, \tilde{t}) = (\sigma(\tilde{s}), t) \in R$. Then also $(\pi(\sigma(\tilde{s})), \pi(t)) = (\pi(\sigma(\tilde{s})), u) \in R$. Let $v = \pi(\sigma(\tilde{s}))$. This state belongs to $\mathit{Orbit}(s, \mathbb{P})$, since $\tilde{s}$ does and $\pi, \sigma \in \langle \mathbb{Q} \rangle \subseteq \langle \mathbb{P} \rangle$. Applying the induction hypothesis to the prefix $\hat{q}$ of $\hat{p}$ of length m, which ends in $\hat{p}^m = (s, \mathbb{P})$, there is a path q in M of length m leading to v. We can build a path in M of length $m+1$ by appending u to q, since $(v, u) \in R$. $\square$

Looking back at figure 3, clearly for every path in M there is a *corresponding* path in $\widehat{M}$, and vice versa, in the sense of lemmas 9 and 10. Note that the concrete states AI and IA, which are collapsed to the abstract state $(AI, |\,1, 2\,|)$, are not bisimilar — even their immediate futures differ. However, the path correspondence lemmas hold and are in the end enough for preserving shortest paths and reachability, which is the objective of this paper.

The path correspondence also sheds light on the path lifting procedure sketched in section 7, "Error Checking". Suppose $\hat{t}$ has been found in line ii of algorithm 1 to represent a concrete error state u. Let $\hat{p}$ be the abstract path leading to $\hat{t}$; path $\hat{p}$ is obtained straight-forward by following abstract back-edges recorded during the exploration. We then have $u \in \mathit{Orbit}(\hat{p}^{|\hat{p}|})$, and lemma 10 applies. The lemma guarantees the existence of a concrete error path p; its proof constructively describes how the edges of p can be found.

8.3. Reachability on $\widehat{M}$ Without Subsumption

Consider now a variant of our lazy algorithm in which the *update* function behaves as in standard reachability analysis: it checks whether the new state $\hat{t}$ already exists in *Reached* and, if not, adds it to *Reached* and to *Unexplored*. This variant is shown in algorithm 2.

Algorithm 2 Lazy Symmetry Reduction without Subsumption

```
Input: initial state $\lambda \in S$
  (lines 1-12 as before)

Subroutine update$(t, \mathbb{Q})$: updating Reached and Unexplored
Input: newly computed state $\hat{t} = (t, \mathbb{Q})$
  i:  if $\hat{t} \notin$ Reached
 ii:      if $\hat{t}$ represents a concrete error state
iii:          return concrete error path and exit
 iv:      add $\hat{t}$ to Reached; enqueue $\hat{t}$ into Unexplored
```

Algorithm 2 repeatedly dequeues an element from *Unexplored*, expands it and simply adds anything new to *Reached* and to *Unexplored* (line iv in algorithm 2). It thus performs breadth-first style reachability analysis on the structure whose edges are defined by the expansion procedure in lines 4–11, which is identical to that in algorithm 1. To prove that algorithm 2 implements reachability analysis on $\widehat{M}$, it remains to show that lines 4–11 generate exactly the successors of state $\hat{s}$ under the transition relation $\widehat{R}$ of $\widehat{M}$ (definition 8).

We first prove that the value computed by function *unwind* satisfies the important equation (5):

Lemma 11. Let $s \in S$, $\mathbb{P}$ and $\mathbb{L}$ be partitions, and let $\mathbb{Q} = glb(\mathbb{P}, \mathbb{L})$. The set $U = \{s\} \cup \{\pi(s) : \pi \in \langle \mathbb{P} \rangle \setminus \langle \mathbb{L} \rangle\}$ satisfies

$$\bigcup_{u \in U} Orbit(u, \mathbb{Q}) = Orbit(s, \mathbb{P}) .$$

Proof:

$\Rightarrow$: Consider $u \in U$ and $\sigma \in \langle \mathbb{Q} \rangle$; we show $\sigma(u) \in Orbit(s, \mathbb{P})$. By the definition of U, there exists $\pi \in \langle \mathbb{P} \rangle$ such that $u = \pi(s)$. Hence $\sigma(u) = \sigma(\pi(s)) \in Orbit(s, \mathbb{P})$ since $\sigma \in \langle \mathbb{Q} \rangle \subseteq \langle \mathbb{P} \rangle$, $\pi \in \langle \mathbb{P} \rangle$.

$\Leftarrow$: Consider $\sigma(s)$ for $\sigma \in \langle \mathbb{P} \rangle$. We distinguish two cases: (i) If $\sigma \in \langle \mathbb{P} \rangle \setminus \langle \mathbb{L} \rangle$, then $\sigma(s) \in U$ (via $\pi := \sigma$), thus $\sigma(s) \in Orbit(\sigma(s), \mathbb{Q}) \subseteq \bigcup_{u \in U} Orbit(u, \mathbb{Q})$. (ii) If $\sigma \in \langle \mathbb{P} \rangle \cap \langle \mathbb{L} \rangle = \langle \mathbb{Q} \rangle$, then $\sigma(s) \in Orbit(s, \mathbb{Q}) \subseteq \bigcup_{u \in U} Orbit(u, \mathbb{Q})$ since $s \in U$. □

We now prove in two lemmas that lines 4–11 generate exactly all successors under $\widehat{R}$ (definition 8).

Lemma 12. State $(t, \mathbb{Q})$ passed to *update* in line 12 is a valid successor of $(s, \mathbb{P})$ under transition relation $\widehat{R}$.

Proof: Consider $\hat{t} := (t, \mathbb{Q})$ with t and $\mathbb{Q}$ as defined in the algorithm. We use definition 8 to show that $((s, \mathbb{P}), (t, \mathbb{Q})) \in \widehat{R}$. Let $\tilde{s} := u$ (line 7), $\tilde{t} := v$ (line 10, *before* canonization), and $\mathbb{L}$ from line 4. Then $\tilde{s} = u \in Orbit(\hat{s})$ by the definition of U (equation (5)), $\tilde{t} = v \in Orbit(v, \mathbb{Q}) = Orbit(t, \mathbb{Q}) = Orbit(\hat{t})$, since canonization does not change orbits. Further, $(\tilde{s}, \tilde{t}) = (u, v) \in R$ via $\mathbb{L}$, since line 9 implements precisely the conditions on R as defined in equation (4) (section 4.2). Finally, $\mathbb{Q} = glb(\mathbb{P}, \mathbb{L})$ by construction (line 5), so $((s, \mathbb{P}), (t, \mathbb{Q})) \in \widehat{R}$ by definition 8. □

Lemma 13. Every successor of $(s, \mathbb{P})$ under transition relation $\widehat{R}$ is eventually passed to *update* in line 12.

Proof: Let $((s, \mathbb{P}), (t, \mathbb{Q})) \in \widehat{R}$ for some $(t, \mathbb{Q})$. Then there exist $\tilde{s} \in Orbit(s, \mathbb{P})$, $\tilde{t} \in Orbit(t, \mathbb{Q})$ and $\mathbb{L}$ such that $(\tilde{s}, \tilde{t}) \in R$ via $\mathbb{L}$ and $\mathbb{Q} = glb(\mathbb{P}, \mathbb{L})$. From $(\tilde{s}, \tilde{t}) \in R$, we conclude that there is an edge e labeled with $\mathbb{L}$ that accounts for this transition in M. Let j be the index of the process component that changes its local state in $(\tilde{s}, \tilde{t})$. Edge e is eventually selected by algorithm 1 in line 4. Line 5 computes $\mathbb{Q} = glb(\mathbb{P}, \mathbb{L})$, equal to the $\mathbb{Q}$ that t is annotated with.

Since $\tilde{s} \in Orbit(s, \mathbb{P})$, by equation (5) there exist an element u such that $\tilde{s} \in Orbit(u, \mathbb{Q})$. Element u is eventually selected in line 7. Since u is identical to $\tilde{s}$ up to a permutation from $\langle \mathbb{Q} \rangle$ and edge e is invariant under permutations from $\langle \mathbb{L} \rangle$ and thus from $\langle \mathbb{Q} \rangle$, it follows that the conditions in line 9 are satisfied for some index i that is identical to j up to a permutation from $\langle \mathbb{Q} \rangle$. Value v computed in line 10 is identical to $\tilde{t}$ and to t from above up to a permutation from $\langle \mathbb{Q} \rangle$. Since $(t, \mathbb{Q})$ is canonical, the canonization in line 11 maps v to t. Thus, $((s, \mathbb{P}), (t, \mathbb{Q}))$ is recognized by algorithm 1 as a transition in $\widehat{M}$. □

As an example, when given the state transition diagram that induces the Kripke structure in figure 3, algorithm 2 generates the abstract states on the right of the same figure. Recall that the structures in figure 3 are not bisimilar. As a consequence, even without subsumption the algorithm presented in this paper cannot preserve bisimulation. This is not a bug, but a feature: *virtual symmetry* [EHT00, see section 11] is the most generous condition permitting a bisimilar quotient structure. The goal of this paper is to increase the coverage of systems (i.e. allow yet more irregularity), while focusing on state space exploration as opposed to full model checking.

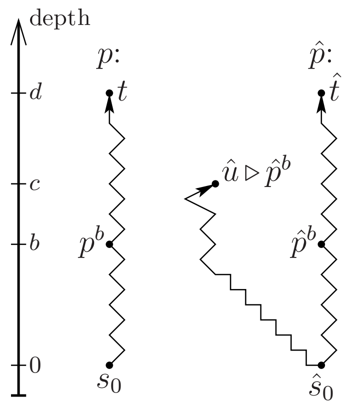
8.4. Reachability on M With Subsumption

We finally show that incorporating subsumption into the procedure, resulting in algorithm 1, gives a routine equivalent to reachability analysis on the original structure M.

Lemma 14. Let $t \in S$. If t is reachable in M with depth d, then algorithm 1 finds a path to an abstract state $\hat{t}$ with $t \in \mathit{Orbit}(\hat{t})$, and the shortest such path it finds has length d.

Proof: We show that algorithm 1 finds such a path of length d. The fact that it does not find a shorter path follows from lemma 10.

Let p be a shortest path in M of length d ending in t (see figure on the right). By lemma 9, there is a corresponding path $\hat{p}$ in $\widehat{M}$ of length d ending in some $\hat{t}$ with $t \in \mathit{Orbit}(\hat{t})$. By the result of section 8.3, algorithm 2 finds $\hat{t}$ with depth d. It remains to show that the pruning due to subsumption in algorithm 1 does not remove any of the states along $\hat{p}$ up to and including $\hat{t}$. Specifically, we show, for any $b \leq d$, that any state $\hat{u}$ reachable in $\widehat{M}$ that subsumes $\hat{p}^b$ (the bth state along $\hat{p}$) has depth at least b and is therefore not reached before reaching $\hat{p}^b$. As a result, path $\hat{p}$ is found by algorithm 1.



To show the claim, assume $\hat{u}$ is reachable with some depth c and $\hat{u} \triangleright \hat{p}^b$; we show $c \geq b$. Let as always p^b be the bth state along p. By the correspondence of p and $\hat{p}$, it is $p^b \in \mathit{Orbit}(\hat{p}^b) \subseteq \mathit{Orbit}(\hat{u})$. By lemma 10, there is a path in M ending in p^b that corresponds to the path to $\hat{u}$; this path has thus length c. Since p is a shortest path to t, every prefix of p is a shortest path to the state it ends in, so p_b has depth b. Therefore, $c \geq b$. □

Lemma 15. Let $t \in S$. If t is not reachable in M, then algorithm 1 finds no $\hat{t}$ with $t \in \mathit{Orbit}(\hat{t})$.

Proof: via the contrapositive. If algorithm 1 finds an element $\hat{t}$ with $t \in \mathit{Orbit}(\hat{t})$, then so does algorithm 2: it is identical to algorithm 1 except that it does not employ subsumption, which may at most prune states. By the result of section 8.3, $\hat{t}$ is reachable in $\widehat{M}$. By lemma 10, there is a path in M to t, so t is reachable in M. □

The following corollary summarizes the relationship between states reachable in M and states found by algorithm 1 in *Reached*.

Corollary 16. Let $t \in S$. State t is reachable in M iff algorithm 1 finds an abstract state that represents t. If so, the depth of t in M equals the length of a shortest path algorithm 1 finds to any such abstract state.

9. Implementation and Efficiency

We discuss essential refinements of algorithm 1 and derive efficiency upper bounds.

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.

A symmetric edge e in line 4 of algorithm 1 allows dramatic simplifications: Lines 5, 6 and 7 can be skipped, as $\mathbb{Q}$ equals $\mathbb{P}$ and U reduces to $\{s\}$. The test $(i,u) \models \phi$ can be factored out of the loop in line 8 (replacing i with 1), since it is independent of i (due to ϕ's symmetry). Almost the same simplifications apply if e is asymmetric but $\mathbb{L}$ is coarser than $\mathbb{P}$ ($\langle\mathbb{L}\rangle \supseteq \langle\mathbb{P}\rangle$, which is easy to test): again $\mathbb{Q}$ equals $\mathbb{P}$ and U reduces to $\{s\}$. The test in line 9 is performed separately for each $Q \in \mathbb{Q}$.

If $\mathbb{L}$ is finer than — or incomparable to — $\mathbb{P}$, we must compute $U = \{s\} \cup \{\pi(s) : \pi \in \langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle\}$. We remark that the set of permutation groups generated by partitions is, although closed under intersection, not closed under set difference: $\langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle$ is not even a group. To compute the elements of $\{\pi(s) : \pi \in \langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle\}$, we can proceed as follows. Since $\langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle \subseteq \langle\mathbb{P}\rangle$, every permutation from $\langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle$ is a product of permutations of elements of the partition cells in $\mathbb{P}$. The procedure below therefore steps through the cells $P \in \mathbb{P}$ and builds the set $Perm_P$ of permutations in $\langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle$ that permute elements from P. The elements of $\{\pi(s) : \pi \in \langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle\}$ are then obtained by composing the permutations $Perm_P$, for all P, and applying the compositions to s.

```
1: $Perm_P := \emptyset$
2: **for all** pairs $(x,y) \in P \times P$ such that $x \not\equiv_{\mathbb{L}} y$ **do**
3:     add to $Perm_P$ all permutations $\pi$ such that $\pi(x) = y$
```

Line 2 ensures that permutation π added in line 3 is not a member of $\mathbb{L}$, by finding two elements x and y from different $\mathbb{L}$-cells. Note that the condition $(x,y) \in P \times P \wedge x \not\equiv_{\mathbb{L}} y$ is unsatisfiable whenever P is completely contained in some cell of $\mathbb{L}$. Such cells P can therefore be skipped. Line 3 can be optimized by exploiting potential redundancy in state s in the form of duplicate local states. Many permutations of $\langle\mathbb{P}\rangle \setminus \langle\mathbb{L}\rangle$ may result in the same state when applied to s. This redundancy can be avoided using *buckets*, i.e. process counters for each local state, separately in each cell of $\mathbb{L}$, and forcing π to change the contents of the buckets.

To make the *update* function efficient, the list *Reached* is sorted such that states with local state vectors that are permutations of each other are adjacent, irrespective of their partitions, for example states of the forms $(AAB, \mathbb{P}_1)$, $(AAB, \mathbb{P}_2)$, $(BAA, \mathbb{P}_3)$. Given the newly reached $\hat{t} = (t, \mathbb{Q})$, we first use binary search to identify the range in which to look for candidates for subsumption, namely the *contiguous* range of states in *Reached* whose local state vectors are permutations of t. The search in line i of algorithm 1 for states subsuming $\hat{t}$ can now be limited to this range. Within the range, however, the search must be performed in a linear fashion. The quest for unexplored states $\hat{w}$ with $\hat{t} \triangleright \hat{w}$ in line iv is also linear, but *Unexplored* is usually much smaller than *Reached*. Finally, to insert $\hat{t}$ into *Reached* in line v we can again take advantage of the previously determined range within *Reached* to which $\hat{t}$ belongs.

We present complexity bounds for the lazy exploration technique. Consider the abstract state space $\widehat{S} = S \times Part_n$, which is much bigger than S.[4] Our algorithm, however, generates only a few of the possible partitions of $\{1, \ldots, n\}$ and prunes the search whenever an unexplored state is subsumed by another encountered state.[5] Comparing the lazy technique to standard symmetry reduction and to *plain* exploration (oblivious to symmetry), the informal goal is to show that

$$complexity(lazy) \quad \leq \quad complexity(standard) \quad < \quad complexity(plain)\,. \tag{6}$$

If the automorphism group of the structure induced by a program is non-trivial, standard symmetry reduction is guaranteed to achieve some compression (barring pathological cases), thus the "<" in equation (6). The meaning of "≤" is that this compression is at least 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 $A \xrightarrow{\phi,\mathbb{L}} B$, there is a cell $L \in \mathbb{L}$ containing both processes. Friendship is an equivalence relation on $\{1, \ldots, n\}$. Each class of friends induces a group of automorphisms of the program's Kripke structure. Friends, therefore, enjoy the following property:

Property 17. Let F be a set of friends, l the number of existing local states. Algorithm 1 reaches at most $\binom{|F|+l-1}{|F|}$ local state tuples over the indices in F.

The quantity in the property equals the number of representative states under standard symmetry reduction

[4] The quantities $|Part_n|$ are known as *Bell numbers* and grow rapidly with n.
[5] For a fixed concrete state $s \in S$, the largest number of abstract states of the form $(s, \mathbb{P})$ such that no two subsume each other is given by the largest anti-chain in the partition lattice of $\{1, \ldots, n\}$; see [Can98] for a discussion.

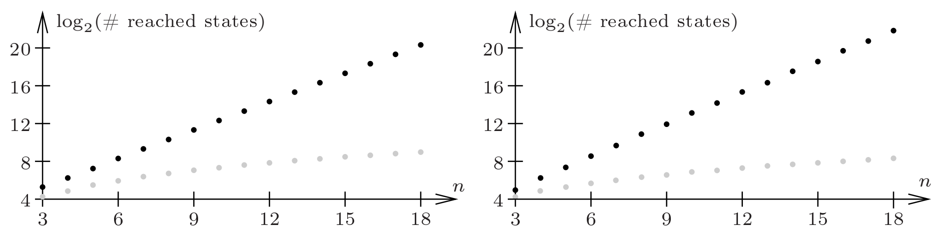


Fig. 4. Comparing the lazy technique (light-gray) to plain exploration (black): reached states for $n/2$ small priority classes $\{1, 2\}, \dots, \{n-1, n\}$ (left) and two large classes $\{1, \dots, n/2\}$ and $\{n/2 + 1, \dots, n\}$ (right)

over the group *Sym* F of all permutations of F and can be calculated using a combinatorial argument: it is equal to the number of combinations (ignoring order) of l objects, taken $|F|$ at a time, with repetition. This quantity can be shown to be at most $l^{|F|}$, and to be much smaller than that if l and $|F|$ get large.

The orthogonal product of all groups of the form *Sym* F, for a class of friends F, is the largest symmetry group that can be derived from the program text. 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 (6) is actually "$<$" or even "$\ll$" depends on the way symmetry is violated and is hard to quantify in general. We observe, however, that for the lazy technique, the notion of *friends* can be extended to include processes not distinguished by edges that are actually *used* during the exploration. Unreachable asymmetric edges reduce the automorphism group, but have no effect on the lazy algorithm. This observation is supported by our experimental results.

10. Experimental Evaluation

We tested the lazy method in a variety of experiments that compare its power to plain state space exploration, to standard symmetry reduction, to lazy exploration *without* subsumption, and to standard symbolic reachability analysis. We borrow a resource controller example from [SG04, p. 729ff.]. The example consists of a server and a family of n clients. There are two communication channels between the server and each client, one for requests and one for replies. Requests cannot be made as long as the server is busy, which is generally the case while it is processing a request; this is recorded in a shared variable. Once idle, the server is ready to receive requests and eventually responds to them.

Interesting for us is that the client processes 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 $\{i : a \leq i \leq z\}$, we label each asymmetric edge in the local state transition diagram with the partition $|\, 1, \dots, a-1 \,|\, a, \dots, z \,|\, z+1, \dots, n \,|$, separating higher, equal and lower priority. A global state's partition annotation can, however, consist of more than three cells.

We fully explore the state space of the system at hand, in order to compare the numbers of reachable states according to various search strategies, and for various values of n. In a first set of experiments, we compare the lazy technique to *plain* exploration, oblivious to symmetry. Figure 4 plots the numbers of reached states over various process counts n on a logarithmic scale. The graphs on the left and on the right differ in the priority scheme, as explained in the caption. 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 enjoys more compression due to larger priority classes.

In a second set of experiments, we compare the lazy technique with standard symmetry reduction, based on the induced structure's automorphism group (figure 5). 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 in the lazy way. The maximum symmetry group is the product of the nine transpositions (1 2) through (17 18), yielding a group size and expected compression factor of only $2^9 = 512$. The actual compression factor $1,310,716/78,729 \approx 17$ is much smaller. This effect is less severe for the less fragmented scheme on the right (thus a linear scale), as is confirmed by the graph.

In a third set of experiments, we investigate the impact of subsumption on the overall algorithm. Recall that subsumption destroys the finite-path correspondence between the original and the implicit abstract structure. If efficiency didn't call for its use, the algorithm would be better off without it. We thus repeated

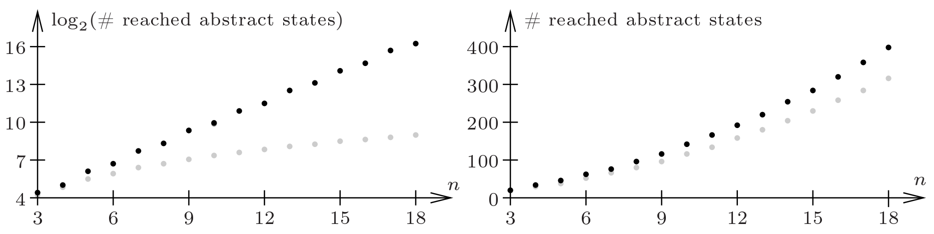


Fig. 5. Comparing the lazy technique (light-gray) to standard symmetry reduction (black); priority schemes as in figure 4

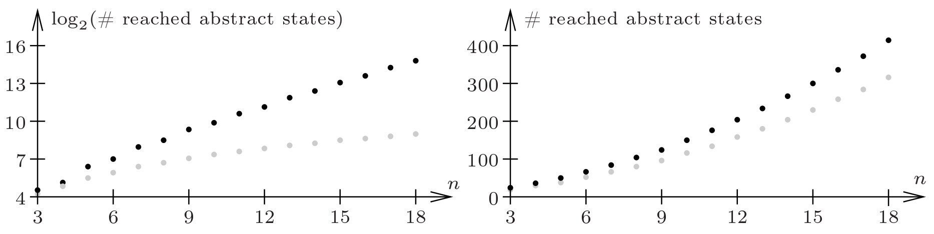


Fig. 6. Comparing the lazy technique with (light-gray) and without (black) subsumption; priority schemes as in figure 4

the runs for the two priority schemes *without* subsumption; the results are shown in figure 6. Comparing this and figure 5, we can see that lazy exploration without subsumption generates a number of states in the same order of magnitude as standard symmetry reduction (black dots in corresponding graphs). The reduction effect achieved by the lazy technique without subsumption is ultimately bounded by what the automorphism group of the state graph permits.

Table 1 shows the physical time and memory consumption for the experiments with $n = 18$. As is recognizable from figure 5, too, standard symmetry reduction performs quite well with the coarse priority scheme; the advantage of the lazy technique here is marginal. In contrast to the number of reachable states reported in the charts above, the time and memory data obviously depend on the machine architecture, which in our case was a 3 GHz Intel$^{\text{TM}}$ Pentium$^{\text{TM}}$ 4 dual-core processor with 2 GB of main memory. Compiler optimizations were not used.

In a fourth set of experiments, we directly investigate how the lazy method scales with increasing fragmentation; the idea to do this is again borrowed from [SG04]. The resource controller example with k priority classes is run with 80 processes; see table 2. In a first variant, denoted "$1, \ldots, 1, \mathit{rest}$", the first $k-1$ classes contain a single process; the final class contains the rest. In a second variant, denoted "$2, \ldots, 2, \mathit{rest}$", the first $k-1$ classes contain two processes; the final class contains the rest. We see from table 2 that the number of reached abstract states grows roughly linearly with k; computation times are very reasonable. For fixed k, the fragmentation grows with the size of the initial $k-1$ classes (1 vs. 2), since these classes reduce the size of the final class, which hosts the majority of the processes.

In a final set of experiments, we compared the lazy technique (with subsumption) against plain, BDD-based *symbolic* state space exploration. The symbolic method computes the set of reachable states as a fixed-point; BDD manipulations are done using the CUDD decision diagram package [Som]. In these experiments, we consider process counts larger than before, since both the lazy and the BDD-based methods can handle them. The results are shown in table 3. As the BDD-based method does not exploit symmetry, the number of reached states is very large and not shown.

We can see that the memory demands of the BDD-based method are higher than for the lazy method, and considerably so for the coarse priority scheme on the right. The BDD-based method is faster only for

Table 1. Time and memory consumption for various techniques ("fine" priority scheme left, "coarse" scheme right); $n = 18$

Technique	Seconds	MBytes	# States
plain expl.	388	104.2	1,310,716
stand. symm.	25	9.1	78,729
lazy w/o subs.	28	27.7	28,261
lazy	1	3.8	505

Technique	Seconds	MBytes	# States
plain expl.	1,445	306.6	3,808,000
stand. symm.	1	3.6	397
lazy w/o subs.	1	3.7	414
lazy	1	3.5	316

Table 2. Lazy symmetry reduction against increasing fragmentation (determined by k); $n = 80$

k	$1,\ldots,1,rest$		$2,\ldots,2,rest$	
	Time	# states	Time	# states
2	1s	558	1s	789
3	2s	792	4s	1245
5	4s	1251	13s	2121
7	8s	1698	24s	2949
10	14s	2346	45s	4101
15	28s	3366	83s	5781
20	44s	4311	118s	7161
25	62s	5181	151s	8241

Table 3. Time and memory consumption, lazy vs. symbolic exploration ("fine" priority scheme: left, "coarse" scheme: right)

Technique	n	Seconds	MBytes	# States
lazy expl.	60	39	12.9	5,461
symbolic	60	8	27.1	—
lazy expl.	70	82	18.5	7,421
symbolic	70	28	29.5	—
lazy expl.	80	158	24.6	9,681
symbolic	80	53	48.2	—

Technique	n	Seconds	MBytes	# States
lazy expl.	60	19	7.5	2,941
symbolic	60	22	28.3	—
lazy expl.	70	39	9.9	3,956
symbolic	70	48	35.7	—
lazy expl.	80	73	13.9	5,121
symbolic	80	87	51.5	—

the scheme on the left, and the advantage shrinks with problem size. We point out the remarkably small number of lazily reached abstract states (a few thousand), despite computation times of up to 3 minutes. We draw two conclusions from these data. First, further efforts to optimize the lazy method should focus on time, rather than memory — the representation is very compact. Second, a symbolic implementation of the lazy technique would likely enjoy the same compactness properties and thus not suffer from the typical BDD memory bottleneck. The success of such an attempt would therefore hinge mostly upon the ability to implement the steps of algorithm 1 time-efficiently using BDDs.

11. Related Work

There are many publications on the use of symmetry for state space exploration and model checking, both of fundamental nature [CEFJ96, ES96] and specific to tools [HBL$^+$03, ID99]. One of the first to apply symmetry reduction strategies to approximately symmetric systems is [ET99]. The authors present the notions of *near* and *rough* symmetry, which are defined with respect to a Kripke structure. In the former case, symmetry violations are allowed only for transitions originating from symmetric states, where all components reside in the same local state. Such transitions can serve as a tie-breaker in applications where priority decides which process gets to enter some exclusive local state first. The second notion is defined using a rather involved concept of coverage among transitions of individual components; it is unclear how to verify coverage on a high-level system description. Examples are limited to versions of the *Readers-Writers* problem.

Near and rough symmetry was generalized in [EHT00] to *virtual symmetry*, the most general condition that allows a bisimilar symmetry quotient. A limitation of this and the preceding approaches is that while bisimilarity makes these approaches suitable for full μ-calculus model checking, it also excludes many asymmetric systems, where bisimilarity to the quotient is simply not attainable. The lazy technique, instead, trades quotient bisimilarity in for coverage of more systems. As with [ET99], it is left open in [EHT00] how the imposed preconditions can be verified efficiently; the techniques presented seem to incur a cost proportional to the size of the unreduced Kripke structure. Our method is adaptive and does not impose any preconditions.

Closest in spirit to our work is that by Sistla and Godefroid [SG04]. As in the present work, they allow arbitrary divergence from symmetry, and account for this divergence initially by conservative optimism, namely in the form of a symmetric super-structure. A *guarded annotated quotient* is then obtained from the super-structure, by marking transitions that were added to achieve symmetry. Loss of precision during the exploration is prevented by means of frequent symmetry checks during runtime. As an advantage, this method can handle arbitrary CTL* properties. Our technique, on the other hand, seems more space-efficient: it applies annotations to states, not edges; in [SG04] there can be multiple annotations to a quotient edge. Further, our method does not require a costly preprocessing of the program text, such as in order to determine a symmetric super-structure. Instead, our method initially lazily ignores the potential lack of symmetry. The idea of one abstract state subsuming another one also appears in [APV06] towards software model checking with the *Java PathFinder*. Critical is the ability to detect subsumption efficiently (theorem 6, in our case).

Symmetry detection solves the problem of suspected but formally unknown symmetry, by inferring structure automorphisms from the program text; recent approaches are given in [DM05, DM06]. An advantage of these approaches is their applicability to quite arbitrary structural symmetries. Our method targets approximately *fully* symmetric systems and might, when confronted with less regular structures, end up annotating each encountered state with the trivial partition consisting of singleton sets. On the other hand, a structure automorphism is global in nature, constraining the entire transition relation. It ignores the possibility of a large part of the state space being unaffected by symmetry breaches. Our lazy symmetry reduction approach detects symmetry (precisely, violations thereof) on-the-fly and can, as such, reduce local substructures with more symmetry than is revealed by global automorphisms.

Our technique bears some resemblance with *lazy abstraction* [HJMS02], which starts from an overapproximation of the concrete model, detects spurious counterexamples, and uses those to locally refine the model. In lazy symmetry reduction, the initial assumption of full symmetry can be viewed as an overapproximation of the actual symmetry group, which is unknown. In both approaches, the idea is to allow the precision of the abstraction to vary across the structure being explored, and to decide on this precision on demand. The difference is *when* the abstraction is refined. In lazy abstraction, this happens when spurious counterexamples have actually been encountered, preventing the target property from being proved. Lazy symmetry reduction is more eager: we don't want to allow spurious paths. Instead, we refine (the symmetry group) as soon as a transition about to be made would introduce imprecision. Our approach therefore remains exact throughout the interleaved exploration and refinement procedure.

12. Summary

We presented a new lazy method for exhaustive state space exploration under partial symmetry. It is intended for, and efficient with, approximately fully symmetric systems, where most local state transitions are available to most processes. Verification of this feature is not required; the method is exact for any input. Besides its broad applicability compared to bisimulation-preserving reduction methods, we believe the algorithm is easy to implement, and incurs justifiable overhead.

The effectiveness of the algorithm rests on two principles: state annotations, in the form of partitions of the set of processes indices, and subsumption. The annotation of a state s tracks how symmetry was violated on the path along which s was encountered; this determines to what extent symmetry may be exploited during future exploration from s. An annotated state subsumes another if its orbit contains that of the other one. As a result of discarding subsumed states, in the end each state is annotated with the *coarsest* partition induced by any path leading to the state. This reduction gives rise to an abstract structure with an equivalent set of reachable states.

We focused in this article on full symmetry, in part because this type is the most frequent and profitable in practice. A natural question for the future is whether the lazy method can be made to work for smaller groups. We first note that for such groups, the benefit of *exact* symmetry reduction diminishes. Non-exact symmetry additionally incurs an (unavoidable) overhead to keep track of the imperfect symmetry. In this paper, we track approximate *full* symmetry in the form of partitions that represent products of fully symmetric subgroups. Partitions are simple enough to permit relatively inexpensive algorithmic manipulations.

Our implementation uses an explicit-state representation. It should be possible to incorporate the algorithm into the model checkers MURφ [ID99] and SPIN [Hol97] and thus extend their scope to asymmetric systems. A symbolic implementation, say using BDDs, appears non-trivial and is one avenue for future work. We have discussed the prospects for such an endeavor near the end of section 10.

Acknowledgements. The authors would like to thank Alastair Donaldson for his insightful comments on an earlier draft of this paper.

References

[APV06] Saswat Anand, Corina Pasareanu, and Willem Visser. Symbolic execution with abstract subsumption checking. In *Model Checking of Software (SPIN)*, 2006.

[Can98] Rodney Canfield. The size of the largest antichain in the partition lattice. *Journal of Combinatorial Theory, Series A*, 1998.

[CEFJ96] Edmund Clarke, Reinhard Enders, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design (FMSD)*, 1996.

[DM05] Alastair Donaldson and Alice Miller. Automatic symmetry detection for model checking using computational group theory. In *Formal Methods (FM)*, 2005.

[DM06] Alastair Donaldson and Alice Miller. Exact and approximate strategies for symmetry reduction in model checking. In *Formal Methods (FM)*, 2006.

[EHT00] Allen Emerson, John Havlicek, and Richard Trefler. Virtual symmetry reduction. In *Logic in Computer Science (LICS)*, 2000.

[ES96] Allen Emerson and Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design (FMSD)*, 1996.

[ET99] Allen Emerson and Richard Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Correct Hardware Design and Verification Methods (CHARME)*, 1999.

[EW03] Allen Emerson and Thomas Wahl. On combining symmetry reduction and symbolic representation for efficient model checking. In *Correct Hardware Design and Verification Methods (CHARME)*, 2003.

[HBL$^+$03] Martijn Hendriks, Gerd Behrmann, Kim Guldstrand Larsen, Peter Niebert, and Frits Vaandrager. Adding symmetry reduction to Uppaal. In *Formal Modelling and Analysis of Timed Systems (FORMATS)*, 2003.

[HJMS02] Thomas Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *Principles of Programming Languages (POPL)*, 2002.

[Hol97] Gerard Holzmann. The model checker SPIN. *Transactions of Software Engineering (TOSE)*, 1997.

[ID99] Norris Ip and David Dill. Verifying systems with replicated components in Murϕ. *Formal Methods in System Design (FMSD)*, 1999.

[SG04] Prasad Sistla and Patrice Godefroid. Symmetry and reduced symmetry in model checking. *Transactions on Programming Languages and Systems (TOPLAS)*, 2004.

[Som] Fabio Somenzi. *The CU Decision Diagram Package, release 2.3.1.* University of Colorado at Boulder, `http://vlsi.colorado.edu/~fabio/CUDD/`.

[Wah07] Thomas Wahl. Adaptive symmetry reduction. In *Computer-Aided Verification (CAV)*, 2007.