Symmetry Reduction and Heuristic Search for Error Detection in Model Checking

Alberto Lluch Lafuente
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
lafuente@informatik.uni-freiburg.de

Abstract

The state explosion problem is the main limitation of model checking. Symmetries in the system being verified can be exploited in order to avoid this problem by defining an equivalence (symmetry) relation on the states of the system, which induces a semantically equivalent quotient system of smaller size. On the other hand, heuristic search algorithms can be applied to improve the bug finding capabilities of model checking. Such algorithms use heuristic functions to guide the exploration. Best-first is used for accelerating the search, while A* guarantees optimal error trails if combined with admissible estimates. We analyze some aspects of combining both approaches, concentrating on the problem of finding the optimal path to the equivalence class of a given error state. Experimental results evaluate our approach.

1 Introduction

Model checking [Clarke *et al.*, 1999] is a formal automated method for the verification of hardware and software systems. Roughly speaking, the state space of the system is explored in order to check whether the error specification holds or not. A negative answer is usually represented by a counterexample that violates the specification. The success of model checking is mainly due to the ability to find and report errors. The main drawback of model checking is the *state explosion problem*, which is especially endemic in asynchronous concurrent systems. In practice, the size of the state space can be large enough to exhaust the available space and time resources. Hence, the main research efforts in this area are focused on mitigating this problem.

Symmetries in a system can be exploited in order to reduce the size of the state space. One way to achieve this is to define an equivalence relation on the states, such that the quotient system induced by the relation is of smaller size while remaining behaviorally equivalent to the original system. Detecting symmetries and deciding whether two states are equivalent or not are the two main practical questions of this approach, whose difficulty also depends on the notion of semantic equivalence required and the class of symmetries exploited. Deciding whether a symmetry relation produces a semantic equivalent system requires, at first, the construction of the full state space. However, practical approaches, like the use of scalarsets [Ip and Dill, 1996], are based on conditions that can be statically checked on the system description. On the other hand, the so-called *orbit problem* which consists of deciding if two states belong to the same equivalence class or orbit, involves -in practice- to find a *canonical* state that represents every state of an orbit. This problem has been shown to be at least as hard as testing graph isomorphism [Clarke *et al.*, 1993]. Heuristics for simplifying the computation of canonical representations have been proposed [Bosnacki *et al.*, 2001; Ip and Dill, 1996]. On the other hand, it is possible to use several *normalized* states to represent an orbit leading to a less complex suboptimal symmetry reduction. For an overview of symmetry reduction approaches we refer to [Ip and Dill, 1996].

The process of verifying a system can be divided in three phases [Cobleigh *et al.*, 2001]. In a first *exploratory* phase one is interested in finding errors quickly. The *fault-finding* phase follows and consists of finding meaningful counterexamples, that can be used to fix the errors. Eventually, the system becomes stable. The *maintenance* phase begins, where one expects successful verification results. The successful ability of model checking to find errors has increased the interest in error detection and report mechanisms. Heuristic search can be applied to improve the first two phases by accelerating the search for errors and providing short (meaningful) counterexamples. We denote this approach as *directed model checking* [Edelkamp *et al.*, 2001a], which have been applied in different domains like, for instance Java Verification [Groce and Visser, 2002], real-time model checking [Behrmann *et al.*, 2001] and data flow analysis [Cobleigh *et al.*, 2001]. The verification is performed by algorithms that apply evaluation functions in order to determine the order in which the states are explored. Algorithms like best-first accelerate the search for errors, while A* provides short or even optimal counterexamples.

In this paper we analyze the combination of symmetry reduction and heuristic search for the detection of safety errors. These two techniques have been already applied together in the planning community [Crawford *et al.*, 1996; Fox and Long, 1999; 2002] and in the CSP community [Gent and Smith, 2000], but to the best of our knowledge this is the first attempt to combine both techniques in system

verification. We concentrate on asynchronous systems and explicit-state verification. Sections 2 and 3 summarize two approaches to symmetry reduction and directed model checking. Section 4 discusses some questions about the combination of both strategies, concentrating on the problem of finding the optimal path to a given error state. Section 5 presents experimental results. The last section concludes the paper and outlines future work.

2 Symmetry Reduction

In the rest of the paper we assume that we are verifying an asynchronous system that involves n identical processes $P_1, \ldots, P_n$. The state space is represented by a labeled transition system T defined as a tuple $\langle S, s_0, \rightarrow, L\rangle$ where S is the set of states, $s_0 \in S$ is the initial state, $\rightarrow \subseteq S \times S$ is the transition relation, and $L : S \rightarrow 2^{AP}$ is a labeling function that maps each state s to the subset of atomic propositions of AP holding in s.

Symmetry Relation. A symmetry of T is a bijection π on S such that $\pi(s_0) = s_0$, and for each pair of states $s, s' \in S$, a transition $s \rightarrow s'$ exists if and only if $\pi(s) \rightarrow \pi(s')$[1].

In practice, symmetries are functions that permute the values of state representations. In the following we use *permutation* and *symmetry* as synonyms. The set of all symmetries forms a group Π with function composition. Any subset Π' of Π generates a subgroup $G(\Pi')$ [2], which, in turn, induces an equivalence (symmetry) relation $\sim$ on the states of S as follows: $s \sim s'$ iff $\pi(s) = \pi(s')$ for some π in $G(\Pi')$.

Any symmetry relation is a congruence on T [Ip and Dill, 1996], which means that for any states $s_1, s_2, s_1' \in S$ such that $s_1 \sim s_2$ and $s_1 \rightarrow s_1'$ there exists a state s_2' and a transition $s_2 \rightarrow s_2'$ such that $s_1' \sim s_2'$. In the following we say that two states s_1, s_2 are *symmetric* iff $s_1 \sim s_2$. Similarly, two transitions $s_1 \rightarrow s_1'$, $s_2 \rightarrow s_2'$ are *symmetric* iff $s_1 \sim s_2$ and $s_1' \sim s_2'$.

Let AP be a set of atomic propositions. We say that AP is invariant under a symmetry relation $\sim$ iff for every pair of symmetric states s_1, s_2 and every atomic proposition $p \in AP$, we have that p holds in s_1 iff p holds in s_2, i.e. $L(s_1) = L(s_2)$. If AP is invariant under $\sim$, then $\sim$ is a bisimulation for AP [Clarke *et al.*, 1999].

Quotient system. A symmetry relation induces a quotient transition system $T_{/\sim} = \langle S_{/\sim}, [s_0], \Rightarrow, L_{/\sim}\rangle$, where $[s]$ denotes the *orbit* or equivalence class of s, and a transition $[s_1] \Rightarrow [s_1']$ exists if there is a transition $s_2 \rightarrow s_2'$ in T such that $s_2 \in [s_1]$ and $s_2' \in [s_1']$. Assuming that the set of atomic propositions AP is invariant under the symmetry relation, uniquely defines the labeling function $L_{/\sim}$ as follows: For each orbit $[s]$, $L_{/\sim}([s]) = L(s')$ for some $s' \in [s]$.

[1] We abbreviate "there exists a transition $s \rightarrow s'$" with "$s \rightarrow s'$".

[2] More precisely $G(\Pi')$ is the closure of Π' under the multiplication and inverse operations.

The Orbit Problem. In practice, symmetry reduction techniques analyze the quotient system $T_{/\sim}$ by exploring a part of T which is constructed on-the-fly. This is done by using states that represent the orbits. A function *rep* : $S \rightarrow S$ associates each state with a representative state of its orbit. A function *rep* is *canonical* iff for every pair of states $s, s' \in S$ we have that $s \sim s'$ exactly when $rep(s) = rep(s')$. In other words, a canonical function provides unique representatives. Canonical functions lead to optimal reductions[3] but can be computationally expensive. Hence, some approaches apply normalizing functions that accept more than one representative for each orbit leading to suboptimal reductions.

Reachability and Bisimulation Equivalence. A symmetry relation preserves reachability [Ip and Dill, 1996]. This means that, given any states $s_1, s_1', s_2 \in S$, if s_1' is reachable from s_1 and $s_1 \sim s_2$, there exists a state $s_2' \sim s_1'$ reachable from s_2. We say that the path from s_1 to s_1' and the one from s_2 to s_2' are *symmetric paths*, if both have the same length and the i-th transitions of each path are symmetric.

As a consequence, a state $s \in S$ is reachable from s_0 in T iff orbit $[s] \in S_{/\sim}$ is reachable from $[s_0]$ in $T_{/\sim}$. Hence, searching for a state $s \in S$ is reduced to looking for $[s] \in S_{/\sim}$ in $T_{/\sim}$. In practice this involves finding a representative of $[s]$ in the explored part of T.

Moreover, a bisimulation symmetry relation induces a bisimulation equivalent quotient system. Hence, specification formulae expressed in temporal logics like CTL or μ-calculus hold in T exactly when they hold in $T_{/\sim}$. Since we focus on the detection of safety errors, this means that if a path violates a safety property, then every symmetric path violates the property. Hence, If a state is an error state then every state in the same orbit is an error state.

3 Directed Model Checking

Contrary to blind search algorithms like depth-first and breadth-first search, heuristic search exploits information of the specific problem being solved in order to guide the exploration of the system. Roughly speaking, algorithms apply functions to establish the desirability of exploring a state. Two of the most frequently used heuristic search algorithms for error detection are A* [Hart *et al.*, 1968] and best-first search [Pearl, 1985]. We now describe a general state expanding search algorithm that can be either instantiated as a depth-first, breadth-first or best-first search algorithm.

General State Expanding Search Algorithm. The general state expanding search algorithm divides the set of states S of a system T into three subsets: the set *open* of visited but not yet expanded states, the set *closed* of visited and expanded states and the rest, i.e. the set of not yet visited states. The algorithm extracts states from *open* and moves them into *closed*. States extracted from *open* are expanded, i.e. their successor states are generated. If a successor of an expanded state is neither in *open* nor in *closed* it is added to *open*.

[3] In the sense that the explored part of T has the same size as $T_{/\sim}$.

Hence, *open* acts as the search horizon. The algorithm terminates if it finds an error state or if there are no more states to be explored.

Breadth-first and depth-first search can be defined as concrete cases of the general algorithm presented above, where the former implements *open* as a queue and the latter as a stack [Pearl, 1985].

Admissibility. A counterexample, error trace or error trail is a path in a system that ends in an error state. Usually, the shorter an error trace is, the easier it is to understand why the error occurred. Sometimes, the most meaningful counterexample is not the shortest one. Hence, our approach allows to assign *costs* to the transitions of the system and aim at finding the counterexample with minimal cost[4], instead of the shortest one. For example, when validating a communication protocol one can assign a cost of 1 to each transition corresponding to a communication operation and cost of 0 to each other transition in order to get the error trace involving the minimal number of communication operations. In the following the cost of a transition $s \rightarrow s'$ will be denoted by *cost*$(s \rightarrow s')$.

An estimate function h maps states into the domain of costs. An estimate is *admissible* if it never over-estimates the cost of reaching the set of goal states starting from a given state. In our context, the set of goal states are those states violating some safety property. Let $h^*(s)$ denote the minimal cost of reaching the set of goal states from u in T. An estimate function h is *admissible* iff $h(s) \leq h^*(s)$ for each state s in T.

An algorithm is *admissible* if it always return the optimal path to the set of goal states. Depth-first search is not admissible, while breadth-first is admissible if the cost of every transition is the same. Dijkstra's single source shortest path algorithm [Cormen *et al.*, 1990] is admissible.

A* and Best-First. Algorithm A* treats *open* as a priority queue in which the priority of a state s is given by function $f(s)$ that is computed as the sum of the cost $s.g$ of the current optimal path from the initial state to s, and the estimated cost $h(s)$ to reach the set of goal states from s. In addition to the general algorithm, A* can move states from *closed* to *open* when they are reached along a path with less cost. This step is called reopening and is necessary to guarantee that the algorithm will find the optimal path to a goal state when non-monotone heuristics are used. In the worst case the number of reopenings is exponential in the size of the state space.

Monotone heuristics satisfy that for each state s and each successor s' of s the difference between $h(s)$ and $h(s')$ is less or equal to the cost of the transition that goes from s to s'. More precisely, an estimate function h is *monotone* iff for every transition $s \rightarrow s'$ of T, $h(s) - h(s') \leq$ *cost*$(s \rightarrow s')$.

A* is admissible if it uses an admissible estimate [Nilsson, 1980]. Every monotone heuristic is indeed admissible [Pearl, 1985]. Moreover, if A* applies a monotone heuristic, reopening is not necessary to preserve admissibility [Pearl, 1985] and the worst case of state expansions remains linear in S.

[4]The cost of a counterexample is the sum of the costs of the transitions composing it.

A* can be applied in the fault-finding phase of the verification process to obtain meaningful counterexamples.

Best-first search manages *open* as a priority queue where the priority of a state is given by an evaluation function f that simply determines the desirability of expanding that node. Best-first is not admissible. However, it tends to find errors fast and is preferred to A* in the exploratory phase of the verification process.

Heuristic Functions. Note that there is an important difference between the exploratory and the fault-finding phase of the verification. In the first phase we don't know if there is an error in the system, while in the second phase we know that there is indeed an error and we have a counterexample obtained in the first phase which can be used to define a heuristic function.

The Hamming distance, for example, is such a function. It is defined as the number of different bits between the bit-vector representation of a state and the given error state. On the other hand, the FSM (Finite State Machine) distance uses the minimum local distances in the state transition graph of the different component processes of the system to derive an estimate [Edelkamp *et al.*, 2001b]. It is of special interest, since it is a monotone admissible heuristic [Lluch-Lafuente, 2003]. Hence, applying it avoids reopening and guarantees that the optimal path to the given error state will be found.

Let $D_i(u, v)$ be the cost of the optimal path from state u to state v in the state transition graph of each process P_i, $pc_i(s)$ the local state (or program counter) of process i in system state s, n the number of process in the system and e the given error state. The FSM distance is defined as $FD^e(s) = \sum_{i=1}^{n} D_i(pc_i(s), pc_i(e))$. Computing this function requires time linear to the number of processes in the system, if D_i is computed before the verification. Computing each matrix D_i can be done in time cubic to the size of P_i [Cormen *et al.*, 1990] which is considerably smaller than the size of the system. In practice, the time required for this pre-computation can be ignored. Note that we have assumed that the n process P_i are identical. Hence, every D_i is equal and in the following we use just D instead of D_i.

Estimating the distance to *unknown* error states is more difficult. In [Edelkamp *et al.*, 2001a] we discuss various alternatives, while the authors of [Groce and Visser, 2002] propose the use of evaluation functions based on thread-interleaving and branch covering metrics.

4 Symmetry Reduction in Directed Model Checking

The algorithm proposed in [Ip and Dill, 1996] to perform an enumerative exploration of a system in the presence of symmetries, is a modification of the general state expanding algorithm which stores a representative of each state instead of the state itself. Hence, one can correctly apply depth-first, breadth-first search or best-first search. Contrary to this kind of algorithms, A* includes reopening. However, we shall see that this is not a hard challenge.

Keeping Track of Paths. In order to be able to produce a counterexample, search algorithms must keep track of the path that leads to each state. Heuristic search algorithms usually associate additional information to each state like the transitions that leads it, usually called *predecessor link*. The path from the initial state to a state s can be constructed by following backwards the predecessor links. If one stores representative states in both the *open* and the *closed* set, the path formed by the predecessor links corresponds to a path in the quotient system but not in the original one, since representative states are not necessarily related by the original transition relation. A solution to avoid this is described in [Bosnacki *et al.*, 2001], which consists of storing representative states in the *closed* set, but not in the *open* set. In addition, the original predecessor link must be maintained, although the corresponding state is transformed into a representative. Figure 1 depicts the algorithm A* with symmetry reduction. Underlined parts are modifications with respect to the original algorithm, which is obtained by replacing *rep(s)* by s. For a state s, $s.g$ represents the current optimal cost of reaching s and $s.f$ the heuristic value given by $s.g + h(s)$, where h is the estimate function. The predecessor link of a state s is stored in $s.t$.

```
( 1)  procedure A*
( 2)    closed ← ∅;  open ← ∅;
( 3)    s0.f ← h(s); s0.g ← 0;  open.insert(s0);
( 4)    while not (open ≠ ∅) do
( 5)      s ← open.extractmin();
( 6)      closed.insert(rep(s));
( 7)      if error(s) then
( 8)        return error path;
( 9)      for each transition s → s' ∈ T do
(10)        f' ← s.g + cost(s → s') + h(s');
(11)        if s' ∈ open and (f' < s'.f) then
(12)          s'.f ← f'; s'.g ← s.g + cost(s → s');
(13)          s.t ← (s → s');
(14)        else if rep(s') ∈ closed and (f' < rep(s').f) then
(15)            s'.f ← f'; s'.g ← s.g + cost(s → s');
(16)            s.t ← (s → s');
(17)            closed.delete(rep(s'));
(18)            open.insert(s');
(19)        else if s' ∉ open and rep(s') ∉ closed then
(20)          s'.f ← f'; s'.g ← s.g + cost(s → s');
(21)          s.t ← (s → s');
(22)          open.insert(s');
```

Figure 1: A* search algorithm with symmetry reduction.

Reopening. With the approach of [Bosnacki *et al.*, 2001] for correctly keeping track of paths, one has to modify the way reopening is applied (Lines 11-18). A* must check whether a state s is in $open$ (Line 11) or *rep*(s) is in *closed* (Line 14). In the latter case and if s has been reached through a path with lower cost, then *rep*(s) and not s must be deleted from *closed* (Line 17). Note that our algorithm is still sound, since the only difference between A* and the general state expanding algorithm in presence of symmetries is that A* can explore a state many times due to the reopening issue. Hence, reachability is still preserved.

Path costs. A question that arises is whether or not the cost of the optimal error path in T is the same as the cost of the optimal error path in the explored part of T as representative of the quotient system $T_{/\sim}$. First, it is easy to see that the shortest path from a state s to a state s' in T has the same length as the shortest path from $[s]$ to $[s']$, since for each path from a state of $[s]$ to a state of $[s']$ there exists a symmetric path from s to s'.

It is also clear that if we assume that symmetric transitions have the same cost then symmetric paths also have the same cost. In the following we assume that this is indeed the case, such that the cost of a transition $[s_1] \Rightarrow [s_1']$ is uniquely defined as the cost of one of the original transitions $s_2 \Rightarrow s_2'$ such that $s_2 \in [s_1]$ and $s_2' \in [s_1']$. As a consequence, the optimal error path in the original and in the quotient system both have the same cost.

Symmetric Heuristic Functions. An additional issue one has to regard is the computation of the heuristic values. We have assumed that for every orbit, the cost of the optimal error path starting from each state of the orbit is the same. It is not guaranteed, however, that every state of an orbit receives the same heuristic value. We say that a heuristic function h is symmetric if for each pair of symmetric states $s, s' \in S$, $h(s) = h(s')$. If a symmetric function h is applied, one can just use $h(s)$ as heuristic value for the orbit of a state s.

The formula-based heuristic [Edelkamp *et al.*, 2001a], is an example of such a function. It takes into account only the values of the set of propositions AP. If AP is invariant under the symmetry relation each pair of symmetric states receives the same value.

Improving Error Trails. We now concentrate on one of the goals of our approach to directed model checking: improving already established counterexamples. We distinguish two subproblems: searching for exactly the same error state, and searching for states violating the same specification.

Searching for a given state e is equivalent to finding a counterexample for a specification requiring that *state e is never reached*. Hence, applying symmetry reduction requires a symmetry relation such that no state is symmetric to e but e itself. Otherwise, the specification would not be symmetric. Obviously, such a symmetry relation is not easy to achieve. Assume, for example, that the program counter of every process in state e is different. Clearly, no pid[5] permutation π but the identity will guarantee that $e = \pi(e)$. Unfortunately, we are precisely exploiting process symmetries which requires to apply this kind of permutations. Hence, another strategy is necessary.

Let f be the specification being verified, for which the path to e represents a counterexample. Applying a symmetry that preserves f, ensures that every state in the orbit of e violates

[5]Process identity.

f. Hence, it is possible to search for $[e]$ in the quotient system. If we do this, we will obtain a path that goes from the initial state to some state e' in the orbit of e. Now, we know that there must be a symmetric path that goes from the initial state to e. This path can be obtained by processing the path to e. Starting from the last transition of the path, say $s' \rightarrow e'$, the last transition of the desired path will be a symmetric transition $s \rightarrow e$ in T. The sub-path to s' is converted into the sub-path to s in the same way.

Searching for an Orbit. In the following, we concentrate on the problem of finding the shortest path to the orbit of a known error state e. First, we discuss how to check whether or not $[e]$ has been found. In other words, if during the exploration we find a state s, how do we check if $[s] = [e]$ [6]? If rep is canonical we can just use the following equivalence $rep(s) = rep(e) \Leftrightarrow [s] = [e]$. If rep is not canonical, the equivalence does not hold and one has to check if s is exactly one of the states in $[e]$. This can be done as follows. Before the verification, we store $[e]$ in a hash table, and during the search we check whether or not s belongs to the hash table in order to determine if $s \in [e]$. Though this can be done in constant time [7], this test is still less efficient than checking $rep(s) = rep(e)$.

Distance to an Orbit. The second issue regards the heuristic functions. The two heuristic functions to shorten counterexamples of our approach, namely the Hamming and the FSM distance, estimate the distance from a state s to the given error state e. But how do we estimate the distance from s to the orbit of e? A naive solution is to use the heuristic value assigned to s or to $rep(s)$. However, this is not a good idea. If during the exploration of $T_{/\sim}$ we encounter a state s, it can happen that the goal error state e is reachable neither from s nor from $rep(s)$, though we know that there exists some state e' in $[e]$ that is reachable from s in the part of T being explored. For example, the FSM distance can deliver the value ∞ which is evidently no lower bound for the actual distance from $[s]$ to $[e]$. This is because FD^e estimates distances in T. We need functions that estimate the distance from $[s]$ to $[e]$ in $T_{/\sim}$. In other words, the naive heuristic is not admissible.

Now, we define such a function based on the existence of a heuristic function h^e estimating the distance to e and prove that if h^e is admissible or monotone, so will be the defined function.

Definition 1 *Given a function h^e that estimates the distance from s to e, we define an estimate that approximates the distance from $[s]$ to $[e]$ as follows: $h^{[e]}_{/\sim}(s) = \min_{e' \in [e]}\{h^{e'}(s)\}$.*

Theorem 1 *If h^e is admissible, then $h^{[e]}_{/\sim}$ is admissible.*

Proof. We have to prove that $h^{[e]}_{/\sim}(s)$ is a lower bound for the cost of reaching $[e]$ from $[s]$ in the quotient system, given that $h^{e'}(s)$ is a lower bound for the cost of reaching a state $e' \in [e]$ from s in the quotient system.

First, we know that at least one of the states of $[e]$ is reachable from s in T. Let e' be any such state. We have assumed that the cost of the optimal path from s to e' in T and the cost of the optimal path from $[s]$ to $[e]$ in $T_{/\sim}$ is the same. $h^e(s)$ is a lower bound for the cost of the first path and by Definition 1 $h^{[e]}_{/\sim}(s)$ is smaller or equal than $h^e(s)$. Hence, $h^{[e]}_{/\sim}(s)$ is a lower bound for the cost of the optimal path from $[s]$ to $[e]$ in $T_{/\sim}$. □

Theorem 2 *If h^e is monotone, then $h^{[e]}_{/\sim}$ is monotone.*

Proof. We have to prove that $h^{[e]}_{/\sim}(s) - h^{[e]}_{/\sim}(s') \leq cost(s \rightarrow s')$ for every transition $s \rightarrow s'$ of T, given that $h^e(s) - h^e(s') \leq cost(s \rightarrow s')$ for every transition $s \rightarrow s'$ of T.

Suppose the contrary, i.e that there exists a transition $s \rightarrow s'$ of T such that $h^{[e]}_{/\sim}(s) - h^{[e]}_{/\sim}(s') > cost(s \rightarrow s')$. Let $h^{[e]}_{/\sim}(s) = h^{e_1}(s)$ and $h^{[e]}_{/\sim}(s') = h^{e_2}(s')$, with $e_1, e_2 \in [e]$. Then, we have $h^{e_1}(s) - h^{e_2}(s') > cost(s \rightarrow s')$. By Definition 1 we know that $h^{e_2}(s) \geq h^{e_1}(s)$ and, therefore, we have $h^{e_2}(s) - h^{e_2}(s') \geq h^{e_1}(s) - h^{e_2}(s') > cost(s \rightarrow s')$ But this is not possible since we have assumed that h^{e_2} is monotone and thus $h^{e_2}(s) - h^{e_2} \leq cost(s \rightarrow s')$. □

The time complexity of $h^{[e]}_{/\sim}$ depends on the time complexity of h^e and the number of states in $[e]$. Note that this number depends on the class of permutations being applied. For example, for the full permutation group of pid symmetries with n processes, an orbit can have $O(n!)$ states. The time required to compute this heuristic can avoid in the worst case the savings offered by heuristic representative computations, which in turn try to avoid computing each state in an orbit.

On the FSM Distance. The above described approach can be applied to the FSM distance in order to devise an admissible estimate $FD^{[e]}_{/\sim}$ for the distance to the orbit of $[e]$. Note that this approach is not always necessary. Clearly, the FSM distance is symmetric if we do not use pid permutations, since in this case the values of each pc_i remains the same for each state in an orbit. However, most symmetries are due to the processes and pid permutations are necessary.

On the other hand it is not necessary to consider every state in $[e]$, but only those with different pc_i values. Note that two symmetric states can have the same pc_i but different values of other variables of the system; they are two different states, but have the same heuristic value.

An alternative to $FD^{[e]}_{/\sim}$ can be to (under)estimate the distance to $[e]$ as follows: $fd^{[e]}_{/\sim}(s) = \sum_{i=1}^{n} \min_{1 \leq j \leq n} D(pc_i(s), pc_j(e))$.

Theorem 3 $\mathrm{fd}^{[e]}_{/\sim}$ *is admissible.*

Proof. We have to proof that for each $s, e \in S$, $fd^{[e]}_{/\sim}(s)$ is a lower bound for the distance from $[s]$ to $[e]$ in $T_{/\sim}$. It is easy to see that $\sum_{i=1}^{n} \min_{1 \leq j \leq n} D(pc_i(s), pc_j(e))$ is smaller than $\sum_{i=1}^{n} D(pc_i(s), pc_i(e)) = FD^{[e]}_{/\sim}(s)$. Since FD never over-

[6] Note that this is indeed the orbit problem.

[7] Using perfect hashing, for example.

estimates the distance from $[s]$ to $[e]$ in $T_{/\sim}$, so does $fd^{[e]}_{/\sim}(s)$. □

Clearly, $fd^{[e]}_{/\sim}$ is less informed than $FD^{[e]}_{/\sim}$. On the other hand, $fd^{[e]}_{/\sim}$ can be computed in time quadratic to the number of processes of the system, which is in general asymptotically better than the time complexity of $FD^{[e]}_{/\sim}$. For example, if full pid permutations are applied, the time required to compute $FD^{[e]}_{/\sim}$ is $O(n! \cdot n)$, since in the worst case $[e]$ has $n!$ many states. If, on the other hand, rotational permutations are applied, both estimates have the same time complexity, i.e. $O(n^2)$.

5 Experiments

We have performed some experiments with our experimental model checker HSF-SPIN [Edelkamp *et al.*, 2001a] in order to evaluate the combination of heuristic search with symmetry reduction, considering the situation where we want to find the shortest path to a given error state, which is provided by a previous verification run. Due to the lack of space we do not include the results of experiments that show that the combination of both techniques is better than one of them in isolation. All results were produced on a SUN workstation, UltraSPARC-II CPU with 248 Mhz. We use a time and space constraint of 10 hours and 512 MB. If an experiment requires more time or space we denote it with o.t. and o.m., respectively.

We use four scalable models as test cases. Two of them are ring protocols exhibiting rotational symmetries, while the rest are protocols in which full symmetries can be exploited.

The first ring protocol is a deadlock solution to the well-known dining philosophers problem, while the second is a model of a leader election algorithm [Dolev *et al.*, 1982]. In the original, correct algorithm each node in the ring has a different number and the process with the highest number is elected as leader. In our version of this algorithm, every node has the same number, which leads to an error situation in which two nodes are elected as leaders.

A model of a database manager [Valmari, 1991] and Peterson's mutual exclusion algorithm [Lynch, 1996] were used in [Bosnacki *et al.*, 2001] to empirically analyze different strategies for computing representatives. The original models are error-free, but we have seeded an error in each of them. In the database manager model we split an atomic action, which provokes a race condition leading to a deadlock. On the other hand, we use an incorrect guard in Peterson's algorithm, such that the mutual exclusion property is violated. We have adapted the implementation of [Bosnacki *et al.*,] to our model checker in order to exploit symmetries in these two models.

We apply A* with three different heuristics, namely FD^e, $FD^{[e]}_{/\sim}$ and $fd^{[e]}$ in order to find the optimal path to the orbit of an already found error state e. The error state is obtained by a verification run that uses a non-admissible search algorithm like depth-first or best-first search, which often deliver non-optimal counterexamples. The goal of our experiments was to evaluate the performance of the different heuristics.

Leader election

	n	8	9	10
FD^e	s	2,106	2,641	3,117
	t	1.6	2.5	3.6
$fd^{[e]}_{/\sim}$	s	77,748	222,595	608,981
	t	3:03.8	11:29.6	1:38:46
$FD^{[e]}_{/\sim}$	s	2,106	2,641	3,117
	t	1.6	2.5	3.6
BFS	s	188,514	632,389	o.m.
	t	9:38.7	43:46.3	o.m.
ADFS	s	o.m.	o.m.	o.m.
	t	o.m.	o.m.	o.m.

Dining philosophers

	n	32	64	128
FD^e	s	407	1,583	6,239
	t	0.4	4.7	1:11.0
$fd^{[e]}_{/\sim}$	s	407	1,583	6,239
	t	0.5	7.0	1:46.2
$FD^{[e]}_{/\sim}$	s	407	1,583	6,239
	t	0.4	4.7	1:11.0
BFS	s	o.m.	o.m.	o.m.
	t	o.m.	o.m.	o.m.
ADFS	s	o.m.	o.m.	o.m.
	t	o.m.	o.m.	o.m.

Table 1: Searching for the optimal path to an orbit in two ring protocols.

	FD^e	$FD^{[e]}_{/\sim}$	$fd^{[e]}_{/\sim}$
Leader (n=10)	$4.957e^{-6}$	$5.137e^{-6}$	$3.012e^{-5}$
Philosophers (n=128)	$5.207e^{-5}$	$6.017e^{-5}$	$4.308e^{-3}$
Peterson (n=6)	$4.170e^{-6}$	$1.322e^{-3}$	$3.282e^{-6}$
Database (n=8)	$2.514e^{-6}$	$2.580e^{-3}$	$5.320e^{-4}$

Table 2: Average time to compute the heuristic function for a state.

We also include experiments with breadth-first search (BFS) and the admissible depth-first search algorithm (ADFS) implemented in the SPIN model checker [8]. These two algorithms are clearly outperformed by A*.

Table 1 depicts the results for the two ring protocols. The table includes the number of states and the time required by the algorithm. The length of the delivered path is not shown, since in all cases optimal paths were found.

A* did not reopen any state in any experiment. On the other hand, reopening is the main cause of bad performance of ADFS.

In the model of the dining philosophers, the number of states visited when using FD^e, $fd^{[e]}_{/\sim}$ and $FD^{[e]}_{/\sim}$ is the same. The reason is that in the error state e each process is in the same local state, hence for each state s we have $FD^e(s) = fd^{[e]}_{/\sim}(s) = FD^{[e]}_{/\sim}(s)$. As a consequence the order in which A*

[8] This algorithm is basically a depth-first search with reopening that continues the search when an error state is found and uses the depth of the last error state found as depth bound.

explores the state space is the same. Note that $fd^{[e]}$ requires more time than the other heuristics. This happens because its time complexity is $O(n^2)$, while both $FD^{[e]}_{/\sim}$ and FD^e require only time linear in n, because there is only one state in $[e]$ in this particular case. Table 2 compares the average time required to compute the heuristic functions.

To the contrary in the leader election algorithm the size of $[e]$ is n. Here, $fd^{[e]}_{/\sim}(s)$ forces A* to visit more states than $FD^{[e]}_{/\sim}(s)$, since it is less informed. In addition, $FD^{[e]}_{/\sim}(s)$ and FD^e explore the same number of states. As a consequence, both $FD^{[e]}_{/\sim}$ and FD^e are significantly faster.

`full` strategy

	n	4	5	6
FD^e	s	6,292	34,268	241,370
	t	3.6	1:02.4	48:00
$fd^{[e]}_{/\sim}$	s	14,965	177,496	o.t.
	t	13.2	9:20.7	o.t.
$FD^{[e]}_{/\sim}$	s	5,134	39,885	455,634
	t	4.0	1:43.7	2:04:34
BFS	s	28,379	350,862	o.t.
	t	38.1	26:18.3	o.t.
ADFS	s	o.t.	o.t.	o.t.
	t	o.t.	o.t.	o.t.

`segmented` strategy

	n	4	5	6
FD^e	s	6,292	34,268	241,370
	t	2.8	22.7	8:07
$fd^{[e]}_{/\sim}$	s	14,965	177,496	2,847,925
	t	11.1	5:10.3	7:58:09
$FD^{[e]}_{/\sim}$	s	5,134	39,885	455,634
	t	4.5	49.8	50:52
BFS	s	28,379	350,862	o.t.
	t	29.5	12:47.2	o.t.
ADFS	s	o.t.	o.t.	o.t.
	t	o.t.	o.t.	o.t.

`sorted` strategy

	n	4	5	6
FD^e	s	14,847	109,974	1,792,518
	t	10.1	4:32.4	6:03:8
$fd^{[e]}_{/\sim}$	s	36,658	652,469	o.m.
	t	42.3	54:25.4	o.m.
$FD^{[e]}_{/\sim}$	s	12,847	163,640	o.t.
	t	10.8	7:58.8	o.t.
BFS	s	74,383	1,301,145	o.t.
	t	2:22	2:04:02	o.t.
ADFS	s	o.t.	o.t.	o.t.
	t	o.t.	o.t.	o.t.

Table 3: Searching for the optimal path to an orbit in Peterson's mutual exclusion algorithm

Tables 3 and 4 respectively show the results for Peterson's mutual exclusion algorithm and the database manager. Again, the cost of the path delivered is not shown, since the optimal one was found in all cases. For example in Peterson's algorithm with 6 processes, the original counterexample has a length of 1,181, while the optimal one has a length of 81.

`full` strategy

	n	6	7	8
FD^e	s	190	689	1,225
	t	1.1	32.9	9:20.2
$fd^{[e]}_{/\sim}$	s	549	641	2,794
	t	3.2	30.3	21:58.2
$FD^{[e]}_{/\sim}$	s	343	499	1,524
	t	2.1	25.8	12:14.5
BFS	s	481	1,224	2,900
	t	3.8	1:21.8	30:40.8
ADFS	s	4,212	24,457	o.t.
	t	37.6	28:46.9	o.t.

`segmented` strategy

	n	6	7	8
FD^e	s	190	689	1,225
	t	0.4	7.9	1:48.0
$fd^{[e]}_{/\sim}$	s	549	641	2,794
	t	1.3	9.8	6:38.0
$FD^{[e]}_{/\sim}$	s	343	499	1,524
	t	0.7	7.4	2:44.7
BFS	s	481	1,224	2,900
	t	1.5	26.6	9:93.7
ADFS	s	4,212	24,457	o.t.
	t	13.7	9:34.8	o.t.

`sorted` strategy

	n	6	7	8
FD^e	s	4,539	14,604	119,524
	t	1.5	14.0	4:54
$fd^{[e]}_{/\sim}$	s	13,963	53,464	641,353
	t	11.3	2:13.2	1:29:39
$FD^{[e]}_{/\sim}$	s	7,788	31,618	259,962
	t	5.4	1:01.5	19:27
BFS	s	13,838	96,653	688,846
	t	10.0	5:48.2	1:30:49
ADFS	s	15,847	116,454	o.t.
	t	53.5	29:22.4	o.t.

Table 4: Searching for the optimal path to an orbit in the database manager model.

We apply three symmetry reduction strategies: `full` is a canonical strategy which computes every possible permutation of a state, `segmented` is canonical strategy which uses a heuristic for a more efficient representative computation, and `sorted` is a normalizing strategy.

Table 5 depicts a comparison of the time required for computing the representative of a state, which shows that `sorted` is the fastest one. Of course, the price to pay is a weaker space reduction. For an extended comparison between these strategies we refer to [Bosnacki *et al.*, ; 2001].

On the other hand, determining whether a state is in the orbit of the error state requires more time if the reduction strategy is a normalizing strategy, like `sorted`. Table 6 compares the time required for computing the test $rep(s) =$

	full	segmented	sorted
Peterson (n=6)	$3.158e^{-3}$	$7.206e^{-5}$	$1.945e^{-7}$
Database (n=8)	$1.301e^{-1}$	$2.640e^{-2}$	$4.554e^{-3}$

Table 5: Average time to compute the representative of a state.

	$rep(s) = rep(e)$	$s \in [e]$
Peterson (n=6)	$4.792e^{-7}$	$1.588e^{-4}$
Database (n=8)	$5.671e^{-7}$	$3.517e^{-4}$

Table 6: Average time to decide whether a state s is in the orbit of the error state e.

$rep(e)$ applicable with canonical strategies and the test $s \in [e]$ for normalizing strategies [9].

Heuristic FD^e offers, by large, the fastest time efficiency and leads to a lower number of visited states when n increases. We assume that this is due to the actual simplicity of the local state transition graphs of the processes, which avoids FD^e to deliver misleading values.

In Peterson's mutual exclusion algorithm, the size of $[e]$ tends to be $n!/2$, while in the model of the database manager $[e]$ tends to $n(n-1)$. This means, that in the first case one should expect $fd^{[e]}_{/\sim}$ to require significantly less time to compute the heuristic values than $FD^{[e]}_{/\sim}$. This can be more clearly seen in Table 2. However, $FD^{[e]}_{/\sim}$ is more informed leading thus to less visited states. As a consequence, among the two admissible heuristics $FD^{[e]}_{/\sim}$ offers the best performance. For example, in a configuration of Peterson's algorithm with 6 processes and when using the strategy `segmented`, $FD^{[e]}_{/\sim}$ requires about 6 times less space and 9 times less time than $fd^{[e]}_{/\sim}$.

6 Conclusions and Future Work

Exploiting symmetries and applying heuristic search mitigates the state explosion problem of automated verification. The first method reduces the state space to be explored to an equivalent smaller one, while the second technique guides the search in the direction of errors. Both techniques are orthogonal and can be combined without drastic changes in the search algorithms, leading to better results than when each technique is applied alone.

Finding optimal (short) counterexamples is an important question, specially for system designers and debuggers. If symmetry reduction is applied, the shortest path to the error states in the reduced and the original state space have the same length. A* guarantees optimal counterexamples only if combined with admissible heuristics. The FSM distance is an admissible estimate that allows to find the optimal path to a given error state. Nevertheless, when applying symmetry reduction one has to aim at finding the orbit of that state. Finding the orbit of a state e requires to check whether each visited state s belongs to the orbit of e. If a canonical representative function rep is used, it suffices to check whether or not $rep(s) = rep(e)$. If normalizing strategies are applied, we have to check whether s is in $[e]$ which a less efficient test.

We have defined two admissible heuristics for the distance to an orbit $[e]$: $FD^{[e]}_{/\sim}$ and $fd^{[e]}_{/\sim}$. The latter is less informed and runs in time quadratic to the number of processes in the system, while the time complexity of the former is linear in $O(n \cdot |[e]|)$. Experimental results show that in practice, the naive non-admissible heuristic which consists of using the heuristic value of one of the states of the orbit as value for an orbit often delivers the optimal path, while requiring less time than admissible heuristics. Nevertheless, to guarantee that the path found is indeed optimal, one has to apply admissible heuristics. After our experiments, $FD^{[e]}_{/\sim}$ seems to be the best choice among the two admissible heuristics, exploring less states while requiring less time.

Recently we have observed that the problem of computing $FD^{[e]}_{/\sim}$ when applying pid symmetries can be reduced to the problem of finding the minimum weight assignment of a bipartite graph [Melhorn and Näher, 1999]. This problem can be solved in $O(n^3)$ which is asymptotically better than $O(n \cdot |[e]|)$, when $O(|[e]|)$ is smaller than n^2. Initial results are promising. As future work we also plan perform experiments combining heuristic search and symmetry reduction with other state space reduction techniques. For example with partial order reduction, which has been already combined with both symmetry reduction [Emerson *et al.*, 1997] and heuristic search [Lluch-Lafuente *et al.*, 2002]. On the other hand, we would like to study the possibility to import symmetry reduction techniques from other domains, like the detection of new symmetries during the search that is applied in planning [Fox and Long, 2002].

[9] It is worth saying, that our current implementation of the hash table storing $[e]$ is not very efficient.

Acknowledgments

The author is supported by DFG grant Ot64/13-2.

References

[Behrmann *et al.*, 2001] G. Behrmann, A. Fehnker, T. Hune, K.G. Larsen, P. Petterson, J. Romijn, and F. W. Vaandrager. Efficient guiding towards cost-optimality in uppaal. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science 2031. Springer, 2001.

[Bosnacki *et al.*,] D. Bosnacki, D. Dams, and L. Holenderski. Symmetric SPIN. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(1).

[Bosnacki *et al.*, 2001] D. Bosnacki, D. Dams, and L. Holenderski. A heuristic for symmetry reductions with scalarsets. In *International Symposium on Formal Methods Europe*, Lecture Notes in Computer Science 2021, pages 518–533. Springer, 2001.

[Clarke *et al.*, 1993] E. Clarke, R. Enders, T Filkorn, and S. Jha. Exploiting symmetry in temporal logic model

checking. *Formal Methods in System Design*, (9):77–104, 1993.

[Clarke *et al.*, 1999] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[Cobleigh *et al.*, 2001] J. M. Cobleigh, L. A. Clarke, and L. J. Osterweil. The right algorithm at the right time: Comparing data flow analysis algorithms for finite state verification. In *23rd International Conference on Software Engineering (ICSE)*, pages 37–46. IEEE Computer Society, 2001.

[Cormen *et al.*, 1990] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Liks, and A. Roy. Symmetry-breaking predicates for search problem. In *Principles of Knowledge Representation and Reasoning*, 1996.

[Dolev *et al.*, 1982] D. Dolev, M. Klawe, and M. Rodeh. An O(n log n) unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, (3):245–260, 1982.

[Edelkamp *et al.*, 2001a] S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Directed model-checking in HSF-SPIN. In *8th International SPIN Workshop on Model Checking Software*, Lecture Notes in Computer Science 2057, pages 57–79. Springer, 2001.

[Edelkamp *et al.*, 2001b] S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Trail-directed model checking. In Scott D. Stoller and Willem Visser, editors, *1st Workshop on Software Model Checking*, volume 55 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001.

[Emerson *et al.*, 1997] E.A. Emerson, S. Jha, and D. Peled. Comibing partial order and symmetry reduction. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science 1217, pages 19–34. Springer, 1997.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *16th International Joint Conferences in Artificial Intelligence*, 1999.

[Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *6th International Conference on AI Planning and Scheduling*, 2002.

[Gent and Smith, 2000] I.P. Gent and B. Smith. Symmetry breaking during search in constraint programming. In *ECAI*, 2000.

[Groce and Visser, 2002] A. Groce and W. Visser. Model checking java programs using structural heuristics. In *International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press, 2002.

[Hart *et al.*, 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on on Systems Science and Cybernetics*, 4:100–107, 1968.

[Ip and Dill, 1996] C. N. Ip and D. L. Dill. Better verification through symmetry. In *Formal Methods in System Design*, volume 9, pages 41–75, 1996.

[Lluch-Lafuente *et al.*, 2002] A. Lluch-Lafuente, S. Leue, and S. Edelkamp. Partial order reduction in directed model checking. In *9th International SPIN Workshop on Model Checking Software*, Lecture Notes in Computer Science 2318, pages 112–127. Springer, 2002.

[Lluch-Lafuente, 2003] A. Lluch-Lafuente. *Directed Search in the Verification of Communication Protocols*. PhD thesis, Institut für Informatik (Albert-Ludwings-Universität), 2003. To appear.

[Lynch, 1996] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[Melhorn and Näher, 1999] K. Melhorn and St. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Co., Palo Alto, California, 1980.

[Pearl, 1985] J. Pearl. *Heuristics*. Addison-Wesley, 1985.

[Valmari, 1991] A. Valmari. Stubborn sets for reduced state space generation. *Advances in Petri Nets*, pages 491–515, 1991.