First Attempts to Combine Symmetry Reduction and Heuristic Search for Error Detection

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. A symmetry relation on the system is defined, which induces a semantically equivalent quotient system of smaller size. On the other hand, heuristic search algorithms can be applied to improve error detection. 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, presenting an admissible estimate that allows A* to provide optimal counterexamples when combined with symmetry reduction. Experimental results are also included.

Technichal Report No. 184, February 12, 2003

1 Introduction

Model checking [4] 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 specification holds or not. A negative answer is usually represented by a counterexample that violates the specification. The main drawback of model checking is the *state explosion problem*. 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 remains 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 symmetric relation produces a semantic equivalent system requires, at first, the construction of the full state space. However, practical approaches, like the use of scalarsets [14], are based on conditions that can be statically checked on the system description. The so-called *orbit problem* consists of deciding if two states belong to the same equivalence class or orbit. This involves in practice to find a *canonical* state that represents every state of an orbit by permuting symmetric data of state representations. This problem has been shown to be at least as hard as testing graph isomorphism [3]. Heuristics like the use of normalized states [2,13] have been proposed. Several *normalized* states can represent an orbit leading to a less complex suboptimal symmetry reduction. For a concise overview of symmetry reduction approaches we refer to [12].

The process of verifying a system can be divided in three phases [5]. In a first *exploratory* phase one is interested in finding early errors quickly. The *fault-finding* phase follows and consists of finding meaningful counterexamples. Eventually, the system becomes relatively 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 techniques for improving error detection. Heuristic search can be applied for improving the first two phases by accelerating the search for errors and providing short (meaningful) counterexamples. This approach is known as *directed model checking* [8]. 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. 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 theoretical questions about the combination of both strategies. Section 5 presents some 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 processes $P_1, \ldots, P_n$. The state space is represented by a transition system T defined as a tuple $\langle S, s_0, \rightarrow \rangle$ where S is the set of states, $s_0 \in S$ is the initial state and $\rightarrow \subseteq S \times S$ is the transition relation.

Symmetry Relation. A symmetry of T is a bijection p on S such that:

1. $p(s_0) = s_0$,
2. if s is an error state then $p(s)$ is an error state, and
3. for each pair of states $s, s' \in S$, a transition $s \rightarrow s'$ exists if and only if $p(s) \rightarrow p(s')$[1].

The of all symmetries forms a group P with function composition. Any subset P' of P including the identity permutation forms a group under composition and inversion, which in his turn induces an equivalence (symmetry) relation $\sim$ on the states of S as follows: $s \sim s'$ iff $p(s) = p(s')$ for some p of the set of symmetries.

Any symmetry relation is a congruence on T [14], 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 $\sim$ iif 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 iif p holds in s_2. Cleary, for any such set of atomic propositions, the symmetry relation is a bisimulation.

Quotient system. A symmetry relation induces a quotient transition system $T/\!\sim = \langle S/\!\sim, [s_0], \Rightarrow \rangle$, 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 $s_2' \in [s_1']$.

The Orbit Problem. In practice, symmetry reduction techniques explore 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[2] 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. A symmetry relation preserves reachability [14]. 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.*

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. Hence, searching an error state $e \in S$ is reduced to looking for $[e] \in S/\!\sim$ in $T/\!\sim$. In practice this involves finding a representative of $[e]$ in the explored part of T.

[1] We abbreviate "there exists a transition $s \rightarrow s'$" with "$s \rightarrow s'$".
[2] In the sense that the explored part of T has the same size as $T/\!\sim$.

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* [11] and best-first search [18]. 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 of Figure 1 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* (Line 6) and moves them into *closed* (Line 7). States extracted from *open* are expanded (Line 10), 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* (Lines 11-12). 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.

```
( 1)  procedure search
( 2)    closed ← ∅;
( 3)    open ← ∅;
( 4)    open.insert(s0);
( 5)    while (open ≠ ∅) do
( 6)      s ← open.extract();
( 7)      closed.insert(s);
( 8)      if error(s) then
( 9)        return error path;
(10)      for each transition s → s' do
(11)        if not s' ∈ closed ∪ open then
(12)          open.insert(s');
```

Fig. 1. A general state expanding search algorithm.

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 [18].

Admissibility. Usually, the shorter an error trace is, the easier it is to understand what it really means. Sometimes, the most meaningful counterexample is not the shortest one. Hence, one can assign *costs* to the transitions of the system and aim at finding the counterexample with minimal cost[3], 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 $h(s, e)$ is *admissible* if it never over-estimates the cost of reaching a goal state e. Let $h^*(s, e)$ denote the minimal cost of reaching state e from u in T.

Definition 1. An estimate function h is *admissible* iff $h(s, e) \leq h^*(s, e)$ for each state s in T.

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

An algorithm is *admissible* if it always return the optimal path to a goal state. 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 [6] is admissible[4].

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 $g(s)$ of the actual path from the initial state to s, and the estimated cost $h(s, e)$ to reach a goal state e 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 the goal state when non-monotone heuristics are used. In a 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 $h(s, e)$ and $h(s', e)$ is less or equal to the cost of the transition that goes from s to s'.

Definition 2. An estimate function h is *monotone* iff for every transition $s \to s'$ of T, $h(s, e) - h(s', e) \leq cost(s \to s')$.

A* is admissible if it uses an admissible estimate [17]. Every monotone heuristic is indeed admissible [18]. Moreover, if A* applies a monotone heuristic, reopening is not necessary [18] to preserve admissibility and the worst-case of state expansions remains linear in S. 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 first phase of the verification processes.

Heuristic Functions. Note that there is an important difference between the first and the second 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 [7]. It is of special interest, since it is a monotone admissible heuristic [15]. 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 process P_i, $pc_i(s)$ be the local state 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 $H_{FSM}(s, e) = \sum_{i=1}^{n} D_i(pc_i(s), pc_i(e))$. Computing this function requires time linear to the number of processes in the system, if D is computed before the verification. Computing each matrix D_i can be done in time cubic to the size of P_i [6] which is considerably smaller than the size of the system. In practice, the time required for this pre-computation can be ignored.

Estimating the distance to *unknown* error states is more difficult. The formula-based heuristic [8] constructs a function that characterizes error states. Given an error formula f and starting from state s, a heuristic function $h_f(s)$ is constructed for estimating the number of transitions needed until a state s' is reached, where $f(s')$ holds. Constructing an error formula for deadlocks is not trivial. In [8] we discuss various alternatives, including formula based approaches, an estimate based on the number of non-blocked processes, and an estimate derived from user-provided characterizations of local control states as deadlock-prone. Structural evaluation functions based on thread-interleaving and branch covering metrics [10] can also be applied.

[4] Evidently, with a slight modification to terminate if a goal is found.

4 Symmetry Reduction in Directed Model Checking

The algorithm proposed in [14] for performing 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 apply depth-first, breadth-first search or best-first search. Similar modifications must be introduced in A*.

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 lead to the state.

If one stores representative states in both the *open* and the *closed* set, the resulting counterexample corresponds to a path in the quotient state space but not in the original one. A solution to avoid this is described in [2], which consists of storing representative states in the closed set, but not in the open set. In addition, the state information must be separated from the transition information. The original transition information must be maintained, although the corresponding state is transformed into a representative. Figure 2 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 actual cost of reaching s and $s.f$ the heuristic value given by $s.g + h(s)$, where h is the estimate function. The transition that leads to each state s is stored in $s.t$.

Reopening. With the approach of [2] for correctly keeping track of paths, one has to modify the way reopening is applied (Lines 13-20). A* must check whether a state s is in *open* (Line 13) or $rep(s)$ is in *closed* (Line 16). 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 19). 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 symmetric paths have the same length.

It is also clear that if we assume that symmetric transitions have the same cost then symmetric paths have also the same cost. In the following we assume that this is indeed the case.

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. The question that arises is which value must be assigned to an orbit. A naive solution consists of using the heuristic value of one of the representatives. However, if we use an estimate that is admissible for the original system, the resulting estimate may not be admissible for the quotient system.

This can occur, for example, if one applies the FSM distance, which is an admissible estimate. Note that this heuristic is used to find the optimal path to a given state e. Evidently, when symmetry reduction is applied one has to aim at finding the orbit of e, which in practice entails searching for a representative of $[e]$. Applying the FSM distance for the exploration of the reduced system as explained before 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. The FSM distance can deliver the value ∞ which is evidently no lower bound for the actual distance from $[s]$ to $[e]$. This is because H_{FSM} estimate distances in T. We need functions that estimate the distance from $[s]$ to $[e]$ in $T/\!\sim$.

```
procedure A*
  closed ← ∅;
  open ← ∅;
  s0.f ← h(s); s0.g ← 0;
  open.insert(s0);
  while not (open ≠ ∅) do
    s ← open.extractmin();
    closed.insert(rep(s));
    if error(s) then
      return error path;
    for each transition s → s' do
      s'.g ← s.g + cost(s → s'); f' ← s'.g + h(s'); s.t ← (s → s');
      if s' ∈ open then
        if (f' < s'.f) then
          s'.f ← f';
      else if rep(s') ∈ closed then
        if (f' < rep(s').f) then
          s'.f ← f';
          closed.delete(rep(s'));
          open.insert(s');
      else
        s'.f ← f';
        open.insert(s');
```

Fig. 2. A* search algorithm with symmetry reduction.

Definition 3. Given a function h that estimates the distance from s to e, the *estimate h under the symmetry relation* $\sim$ approximates the distance from $[s]$ to $[e]$ and is defined as: $h/\sim(s,e) = \min_{e' \in [e]}\{h(s,e')\}$.

Theorem 4. *If h is admissible, then $h/\sim$ is admissible.*

Proof. We have to prove that $h/\sim(s,e)$ is a lower bound for the cost of reaching $[e]$ from $[s]$ in the quotient system, given that $h(s,e')$ is a lower bound for the cost of reaching each 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. $h(s,e')$ is a lower bound for the cost of the first path and by Definition $h/\sim(s,e)$ is smaller or equal than $h(s,e')$. Hence, $h/\sim(s,e)$ is a lower bound for the cost of the optimal path from $[s]$ to $[e]$ in $T/\sim$.

Theorem 5. *If h is monotone, then $h/\sim$ is monotone.*

Proof. We have to prove that $h/\sim(s,e) - h/\sim(s',e) \leq cost(s \to s')$ for every transition $s \to s'$ of T, given that $h(s,e) - h(s',e) \leq cost(s \to s')$ for every transition $s \to s'$ of T.

Suppose the contrary, i.e that there exists a transition $s \to s'$ of T such that $h/\sim(s,e) - cost(s \to s') > h/\sim(s',e)$. Let $h/\sim(s,e) = h(s,e_1)$ and $h/\sim(s',e) = h(s',e_2)$, with $e_1, e_2 \in [e]$. Then, $h(s,e_1) - cost(s \to s') > h(s',e_2)$. By Definition 3 we know that $h(s,e_2) \geq h/\sim(s,e)$ and, therefore, we have $h(s,e_2) - cost(s \to s') \geq h(s,e_1) - cost(s \to s') > h(s',e_2)$. But this is impossible since we have assumed that h is monotone and thus $h(s,e_2) - cost(s \to s') \leq h(s',e_2)$.

The time complexity of $h/\sim$ depends on the time complexity of h and the class of permutations being applied. For example, for the full permutation group of symmetries on a scalarset of size n, computing each state in an orbit requires $O(n!)$ time. At first, this is independent from the actual number of states in $[e]$. Note that in some cases there exists one unique error state e, or at least less states than possible permutations. Therefore, one can compute the states of $[e]$ first to reduce computational effort. In the worst case, applying heuristics that avoid computing each state in the orbit of a state for finding a representative does not help, since one has to compute for each state the estimate function, which implies computing every state in $[e]$.

5 Experimental Results

We have performed some experiments with our directed model checker HSF-SPIN[5] [8] in order to have a first evaluation of the combination of heuristic search with symmetry reduction, considering the simple situation in which we want to find the shortest path to a given error state.

All results were produced on a SUN workstation, UltraSPARC-II CPU with 248 Mhz. Our test case is a model of a database manager [19] which was used in [2] to empirically analyze different strategies for computing representatives. The original model is error-free, but we have seeded a deadlock. We apply breadth-first search (BFS), A* and symmetry reduction (SYM). Recall that A* with a blind heuristic, i.e. one that assigns the same value to each state, performs BFS. The heuristic used is the FSM distance to an error state previoulsy found by a exploration with depth-first search, which delivers a suboptimal counterexample. Full pid (process identifier) permutations are applied for symmetry reduction using the *full strategy* described in [2].

Table 1 depicts the results for different configurations of the model, where N is the number of database managers. Stored states, and time consumption (hours:minutes:seconds.milliseconds) are included. Time and space of the experiments are limited to 24 hours and 512 MB, respectively.

N		BFS	BFS+SYM	A*	A*+SYM
5	states	12,481	288	1,712	117
	time	4.4	0.2	0.3	0.2
6	states	144,748	791	21,894	415
	time	5:7	2.8	8.6	3.8
7	states	1,600,520	1,867	112,043	537
	time	2:54:59	55.0	1:21	43.3
8	states	out of space	5,337	1,625,414	2,042
	time		24:22	3:48:32	26:6
9	states	out of space	8,273	out of time	2,167
	time		6:47:3		5:50:54
10	states	out of space	out of time	out of time	out of time
	time				

Table 1. Error detection with heuristic search and symmetry reduction.

Time and space reduction is always achieved by A* and SYM in comparison with BFS. Note that time reduction is smaller than space reduction because of the overhead introduced by the computation of the heuristic and representative functions. Recall that given a model with N processes, there are $N!$ possible permutations. For example, when $N = 9$ for every state we have to compute each of the 9! states in its orbit, and hence the long (about 6 hours) computation times of A* and BFS when combined with SYM. In some cases, A* needs more time than BFS. This is because A* requires more time per state for computing the estimate values.

The size of the state space explored when A* and SYM are applied simultaneously is always smaller than when each technique is applied alone, which indicates a good coexistence. However,

[5] Available at `www.informatik.uni-freiburg.de/~lafuente/hsf-spin`

time is not always reduced. In fact, the combination of A* and symmetry reduction requires in some cases more time than BFS with symmetry reduction.

Of course this is just a single example. More experiments exploiting the wide spectrum of symmetry reduction and heuristic search strategies must be performed.

6 Conclusions and Future Work

Exploiting symmetries and applying heuristic search mitigate 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 techniques guides the search in the direction of errors. Both techniques can be combined without drastic changes in the search algorithms.

Finding short counterexamples is an important question, specially for system designers and debuggers. If symmetry reduction is applied, the shortest paths in the reduced and the original state space have the same length. This does not hold if one assigns costs to transitions and aims at optimal-cost counterexamples. In practice, however, the cost of symmetric transitions is the same, such that the optimal counterexamples in the reduced and in the original system have the same cost.

A* guarantees optimal counterexamples only if combined with admissible heuristics. However, finding functions that underestimate the cost or distance to an error state is not easy. We have show how to define a heuristic for the reduced system based on a heuristic for the original one. If the original estimate is admissible or monotone so is the new one. The complexity of the new heuristic depends on the class of symmetric permutations applied and is more suited for canonical reduction strategies.

Our first experimental results show a good coexistence. Since each technique exploits different properties of the system, the combination offers a better space reduction than the single application of one of the techniques. Nevertheless, time reduction is not always achieved in the combination due to the overhead introduced by the computation of representatives and estimate values.

We plan to integrate the approach described in [1] in our experimental model checker. This will allow us to perform more experiments exploiting a wide variety of symmetry reduction and heuristic search strategies. On the other hand, partial order reduction has been combined with both symmetry [9] and heuristic search [16]. We plan to perform experiments combining the three techniques.

References

1. D. Bosnacki, D. Dams, and L. Holenderski. Symmetric SPIN. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(1).
2. 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.
3. 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.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
5. 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.
6. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
7. S. Edelkamp, A. L. Lafuente, and S. Leue. Trail-directed model checking. In S. D. Stoller and W. Visser, editors, *Electronic Notes in Theoretical Computer Science*, volume 55. Elsevier Science Publishers, 2001.
8. 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.

9. E. 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.
10. A. Groce and W. Visser. Model checking java programs using structural heuristics. In *International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press, 2002.
11. 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.
12. C. N. Ip. *State Reduction Methods for Automatic Formal Verification.* PhD thesis, Department of Computer Science of Standford University, 1996.
13. C. N. Ip and D. L. Dill. Better verification through symmetry. In *Conference on Computer Hardware Description Languages and their Applications*, 1993.
14. C. N. Ip and D. L. Dill. Better verification through symmetry. In *Formal Methods in System Design*, volume 9, pages 41–75, 1996.
15. A. Lluch-Lafuente. *Directed Search in the Verification of Communication Protocols.* PhD thesis, Institut für Informatik (Albert-Ludwings-Universität), 2003. To appear.
16. 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.
17. N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Co., Palo Alto, California, 1980.
18. J. Pearl. *Heuristics.* Addison-Wesley, 1985.
19. A. Valmari. Stubborn sets for reduced state space generation. *Advances in Petri Nets*, pages 491–515, 1991.