The Journal of Logic and Algebraic Programming 78 (2009) 274–287

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Search algorithms for automated validation

Tom A.N. Engels, Jan Friso Groote, Muck J. van Weerdenburg, Tim A.C. Willemse [*]

Design and Analysis of Systems Group, Eindhoven University of Technology, The Netherlands

A R T I C L E I N F O

Available online 30 January 2009

Keywords:
Validation
Search algorithms
Random search
Model checking

A B S T R A C T

A novel search technique called *highway search* is introduced. The search technique relies on a *highway simulation* which takes several homogeneous walks through a (possibly infinite) state space. Furthermore, we provide a memory-efficient algorithm that approximates a highway search and we prove that, under particular conditions, they coincide. The effectiveness of highway search is compared to two mainstream search techniques, viz. *random search* and *randomised depth-first search*. Our results demonstrate that randomised depth-first search explores the least amount of states in the effort of finding states of interest, whereas a highways search yields the shortest witnessing traces to such states.

1. Introduction

Random simulation is used in many engineering disciplines as a technique to validate the correctness of a design and get acquainted with its properties. Its popularity is explained largely from the fact that it is push-button technology, limiting user input to a bare minimum. Adding to its popularity are the appeal to intuition that is behind the method, and the sense that a random simulation reflects the typical behaviours of a system.

While random simulation is a proper tool for some designs, its use in a non-deterministic setting is certainly less obvious. This is because the results that are obtained by means of a random simulation are often a poor reflection of the overall behaviour of the design. This is amplified in non-deterministic designs with huge or even infinite state spaces, in which a single simulation visits an insignificant part of the entire state space. In fact, Pelánek et al. [15] demonstrated that for random simulation, the frequency of visits of states has the power law distribution rather than a uniform distribution.

Still, random simulation is often useful when the primary verification tools such as model-checking tools (see e.g. [9]) and theorem proving are incapable of dealing with the problem at hand due to the infamous phenomenon known as "state-space explosion". In such cases, random simulation can be used to search for particularly interesting states or events, leading to a *random search*.

In this paper, we demonstrate that there is ample room for improvement in random simulation. We investigate a technique called *highway simulation* (with the associated search technique called *highway search*), which can be implemented straightforwardly. Furthermore, we provide an algorithm for conducting an approximation of a highway simulation, and we study its properties. Experiments indicate that in practice, an approximate highway search and an ideal highway search do not differ significantly.

Highway simulation is akin to a *restricted breadth-first exploration* [18], which provides a graceful degradation from breadth-first exploration to random simulation, and *parallel random simulation* [15]. The simulation technique (and its associated search technique) remains push-button technology, requiring little to no human intervention. Note that this

[*] Corresponding author.
E-mail address: T.A.C.Willemse@tue.nl (T.A.C. Willemse).

is different from the approach known as *directed model checking*, which uses heuristics to guide a search towards states of interest; using or defining effective heuristics can be very labour-intensive and may require more knowledge of a state space than is available.

The effectiveness of highway search is compared with standard search methods such as *randomised depth-first search* and random search. Randomised depth-first search has been advocated as a yardstick for comparing the efficacy of directed model checking algorithms [17]. However, we feel that the comparison of [17] is one-sided, as it only looks at the probability of finding states of interest in a state space. We believe it is equally important, if not more important, to find *witnesses* (traces through the state space that lead to the state of interest) that are short, as these provide essential feedback to the designers of a misbehaving system. This is not taken into account in [17]. The experiments that we conducted not only consider the amount of states that are required for finding a state of interest in a given state space, but also the length of the witness that leads to the state. Our results show that in nearly all cases, a highway search is the preferred search method if one wishes to find short witnesses, whereas a randomised depth-first search is a viable alternative if one has severe restrictions on available memory.

This paper is structured as follows. Related work and a positioning of our work with respect to related work is discussed in Section 2. Section 3 provides a brief exposition of our formal framework which we use to explain the simulation and search methods. In Section 4 the search methods are introduced and in Section 5 we describe several experiments that we used to study the various search techniques. We present our conclusions in Section 6.

2. Related work

A recent overview of different search methods is provided in [15]. The authors show by means of experiments that the probability of visiting a state in an arbitrary state space using a random simulation has the power law distribution. This means that a random simulation "spends most of the time repeatedly visiting just a few states" [15]. Most of our studies indicate that this indeed hampers the search; however, the classical *dining philosopher* problem (studied in Section 5.2.3) illustrates that this property is useful at times.

Over the years, a variety of search techniques have been proposed; these often take advantage of specific (hardware) architectures. We categorise the search techniques by the following two criteria:

- *distribution*, i.e., the search technique is tailored towards a distributed implementation on, e.g., clusters of processors;
- *guiding*, i.e., the search technique requires heuristics of some sort (either human or rule-based), next to the search objective and the specification, to perform the search.

While it is to be expected that the best (i.e. fastest and shortest) results can be obtained using distributed and guided algorithms, such algorithms also tend to require most resources in terms of hardware and time that has to be invested in providing the right set of heuristics for guiding the search. In general, one may not have these resources available, which is why one often resorts to classic non-distributed non-guided search methodologies, which are *push-button* and use *off-the-shelf* technology. Understanding which search methodology is generally most suited is therefore crucial.

The search methods studied in this paper are *random search*, *randomised depth-first search* [17] and *highway search*, all of which are non-distributed and non-guided search algorithms. Following [17], randomised depth-first search is used as a yardstick for comparing the efficacy of highway search and random search. However, contrary to [17], we refrain from assigning measures to the complexity of our experiments, since the measure of [17] is not independent of the hardware and software running the search algorithms. Instead, we look at the (average) achievements of the searches given a (per experiment) realistic upper bound on the number of states that may be explored.

Distributed search methods. The benefits of distributing a search over several clients are primarily the speed gains and an increased scale of the state space. However, such a gain is only achieved if the searches in all clients are largely disjoint and the overhead in keeping a global view of the search that is conducted is negligible.

In [10], Jones and Sorber describe a method that relies on a parallel search for LTL violations. The search algorithm, called *bee-based error exploration* (BEE) is a fault-tolerant algorithm that is tailored to operate in a general purpose distributed computing environment consisting of workstations that can be switched off at any point in time. The search method itself consists of coordinated depth-bounded random walks. Several examples show that the BEE algorithm outperforms non-distributed model checking algorithms relying on bitstate hashing. The authors suggest the approach might be improved by replacing the random search by better search techniques.

Instead, Sivaraj and Gopalakrishnan [18] assume a reliable network of clients, which they employ to run algorithms that rely on combinations of random walks with bounded breadth-first search. Breadth-first search is used to achieve a wide coverage, whereas random walk is used to find "deep" errors. The approach advocated by Rasmussen et al. [16] can be seen as an improvement over [18]. Their approach relies on specialised search agents that can reside in one or more clients. Agents can trigger searches in other agents running other search algorithms to assist in the exploration when fruitful parts of the state space are found.

A naive, but sometimes successful approach to distribution is to rely on randomness to achieve an even distribution of the work-load among all clients. This approach is taken in e.g. [3], where several clients run a partial randomised depth-first search in parallel. Note that such an approach to distribution applies to most randomised search methods.

Directed search methods. By guiding a search using additional heuristics, one hopes to direct the search effort to the bug-containing part of the state space without wasting time in searching the bug-free part of the state space.

In [6], Groce and Visser propose to use heuristics for *best-first search* and *beam-search*. They provide heuristics that exploit, among others, structural coverage and concurrency structures. The authors reason that the tester or developer of a system has knowledge of the system and accordingly understands where errors are likely to hide. In order to employ such knowledge, their method is open to user-defined heuristics. Three examples, including the dining philosophers, have been conducted using the software model checking tool Java PathFinder [20]. These experiments demonstrate that error finding becomes tractable in some systems where unguided searches do not work very well. However, the authors also rightfully note that their experiments and experience do not suggest a single heuristic to solve the search problem. A different set of heuristics for beam search is further studied by Wijs and Lisser [21], where, moreover, a distributed approach to beam search is taken. Searching for error states in real-time systems can be done using e.g. UPPAAL/DMC [11], which uses a heuristic function to determine the distance to the nearest error state, and Mcta [12], which is based on iterative refinements.

Godefroid and Kurshid [5] introduce a search technique based on *genetic algorithms*. Fitness functions (heuristics) that are tailored to the property that is searched for are used for mutation and selection. The authors compare their techniques to the effectiveness of random search. Even though distribution of the algorithm is not mentioned in [5], it is likely that the mutation and selection processes can be distributed.

3. Framework

We present our techniques in the setting of Transition Systems (TSs). We stress that the notions that we subsequently develop are easily recast into richer frameworks such as *labelled transition systems*.

Definition 1. A *transition system* is a three-tuple $\langle S, s_0, \rightarrow \rangle$, where S is a possibly infinite set of states, referred to as the *state space*, $s_0 \in S$ is the initial state and $\rightarrow \subseteq S \times S$ is the transition relation. We write $s \rightarrow t$ instead of $(s,t) \in \rightarrow$.

Throughout this section, we assume an arbitrary TS $\mathcal{S} = \langle S, s_0, \rightarrow \rangle$. Let $s \in S$ be an arbitrary state. The set of *successor states* of s is defined as $\mathsf{succ}(s) = \{s' | s \rightarrow s'\}$; generalised to a set of states S', we have $\mathsf{succ}(S') = \bigcup_{s' \in S'} \mathsf{succ}(s')$. The TSs that we consider in this paper are *finitely branching*, i.e. for each state $s \in S$, we require $|\mathsf{succ}(s)| \in \mathbb{N}$. A *path* starting in s is a sequence of states $\sigma \equiv t_0\ t_1\ \cdots\ t_n$ for $n \geq 0$ satisfying $t_0 = s$ and $t_{i+1} \in \mathsf{succ}(t_i)$. The state s is *reachable* iff there is a path starting in s_0 ending in s; the set of reachable states of a finite-state TS can be found by means of an exhaustive breadth-first or depth-first exploration (randomised depth-first exploration being a variation on depth-first exploration in which a successor state is selected on a random basis rather than following a pre-determined deterministic strategy). For TSs with an infinite state space, or a finite but extremely large state space, an exhaustive exploration is not feasible because of memory and/or time restrictions. Therefore, some approaches resort to *probabilistic methods*. The most intuitive and straightforward probabilistic state-space exploration method is that of random simulation.

Definition 2. A *random simulation* of the state space is a path σ starting in s_0, for which each s_{i+1} is obtained by a random draw from the set $\mathsf{succ}(s_i)$.

Random simulation suffers from the problem of a poor coverage of the state space (see e.g. [18,15]): in the presence of loops and non-determinism, a random simulation may exhibit revisits of states, leading to a power law distribution for the frequency of state visits. Clearly, unnecessary revisits of states of a state space do not add to the coverage of the state space, even though such revisits sometimes have a surprisingly rewarding effect (see Section 5.2.3). Methods such as randomised depth-first exploration visit each state only once, using back-tracking to continue the exploration upon visiting a state with successor states that have all been explored.

4. Random and highway search

We first set out to define what we mean by a search. A search is based on an underlying exploration or simulation of a state space and requires a search objective in terms of a set of states of interest. Given a path σ, we denote the set of *visited states* of σ by $V(\sigma)$, i.e. $V(\sigma)$ contains all states that appear in σ. Random search, randomised depth-first search and highway search consider all successor states in choosing transitions. The set of all *considered states* of a path σ is denoted by $C(\sigma)$, where for all paths σ' and states s, $C(\sigma' \cdot s) = V(\sigma' \cdot s) \cup \mathsf{succ}(V(\sigma'))$. Note that the successors of the final state of a simulation are not considered as no transition needs to be chosen from such a state.

Definition 3. The result of a *random search* for a set of states $T \subseteq S$ is defined as the set of states $T \cap C(\sigma)$ that is obtained by means of a single random simulation σ.

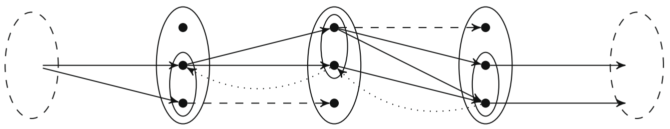


Fig. 1. Typical paths through a transition system.

Clearly, a random search has drawbacks that can be linked to the drawbacks of the random simulation, viz. the frequent revisits of states. Thus, random search as a tool for validating a design seems to be far from optimal. We therefore focus on a hybrid breed of *breadth-first exploration* and random simulation. The resulting simulation method is dubbed *highway simulation*.

Definition 4. Let S and T be arbitrary sets and let $N > 0$ be an arbitrary natural number. We say that T is *N-close to S* iff $T \subseteq S$ and $|T| = \min(N, |S|)$.

Next, we generalise the notion of visited states as follows. Let $\vec{Q} \in (2^S)^*$ be a sequence of sets of states. The set of visited states of $\vec{Q}$ is denoted $V(\vec{Q})$, and is defined as the union of all Q that occur in $\vec{Q}$. The set of considered states, denoted $C(\vec{Q})$, where for all sequences of sets of states $\vec{Q}$ and sets of states Q, $C(\vec{Q} \cdot Q) = V(\vec{Q} \cdot Q) \cup \mathsf{succ}(V(\vec{Q}))$.

Definition 5. A *highway simulation* of width $N > 0$ is defined as a sequence of sets of states $\vec{Q} \in (2^S)^*$ where $\vec{Q} = Q_0 \ldots Q_d$ satisfies:

- $Q_0 = \{s_0\}$ and
- each Q_j (for $j > 0$) is obtained by a random draw from the set P_j:

$$P_j := \left\{ S' | S' \text{ is N-close to } \mathrm{succ}(\mathrm{Q_{j-1}}) \Big\backslash \bigcup_{\mathrm{k<j}} \mathrm{Q_k} \right\}$$

The result of a *highway search* of width $N > 0$ for a set of states T is defined as the set of states $T \cap C(\vec{Q})$ that is obtained by means of a single highway simulation $\vec{Q}$ of width N.

Intuitively, a highway simulation simultaneously explores several *lanes* (set of paths, in our formal terminology), which at each instance can merge or split into new lanes; *exits* (outgoing transition of a state) of a lane are only considered if these lead to new *places* (states, in our formal terminology). Note that the latter requirement may lead to the (premature) termination of a path. The *highway* can be thought of as a collection of lanes. Fig. 1 visualises a typical highway simulation of width 2. The places are indicated by bullets, and the lanes are represented by the small ovals. The large ovals visualise the reachable places from a preceding lane. The exits that are ignored are printed by a dotted line (such exits lead to already visited places and are therefore not *eligible*) or a dashed line (these exits are ignored by chance), whereas the actually taken exits are represented by a solid line.

Note that a highway simulation of width 1 is akin to a random simulation which is not allowed to revisit states, i.e. it will terminate when it cannot avoid closing a loop back to a state it has already visited. This is unlike a random search, which may revisit the state and, at a later stage, may diverge to reach new states.

Property 6. *Let $\vec{Q} = Q_0 \cdots Q_d$ be a highway simulation of width $N > 0$.*

- *For all i, j, $(i \neq j)$, $Q_i \cap Q_j = \emptyset$,*
- *The number of paths starting in s_0, implicitly defined by $\vec{Q}$ is bound from below by $\sum_{i=1}^{d} |Q_i|$.*

Highway simulation (and, hence also a highway search) can be implemented straightforwardly. Such an implementation essentially requires:

(1) Memorising the places that have been visited along a lane,
(2) Computing the set of eligible reachable places from the current lanes and taking a homogeneous N-close selection of this set.

In case of a large highway width N and a large branching degree of the places in the current lane, first computing the eligible reachable places in step 2, and subsequently taking a selection of these places can draw significantly on memory resources and computation times.

The noted problem can be avoided using a probabilistic construction for computing the next set of places in an *on-the-fly* fashion. Algorithm 1 implements this solution. Theorem 7 presented below formalises the assumptions that guarantee that Algorithm 1 correctly implements a highway simulation.

Algorithm 1 Approximate highway simulation

```
1:  $V, d, Q_0 = \{s_0\}, 0, \{s_0\}$;
2:  while $Q_d \neq \emptyset$ do
3:      $c, Q_{d+1} = 0, \emptyset$;
4:      for $s \in Q_d$ do
5:          for $t \in \mathsf{succ}(s)$ do
6:              if $t \notin V \cup Q_{d+1}$ then
7:                  $c = c + 1$;
8:                  if $c \leq N$ then $Q_{d+1} = Q_{d+1} \cup \{t\}$;
9:                  else with probability $\frac{N}{c}$
10:                     select randomly $u \in Q_{d+1}$;
11:                     $Q_{d+1} = (Q_{d+1} \setminus \{u\}) \cup \{t\}$;
12:                 end if
13:             end if
14:         end for
15:     end for
16:     $V, d = V \cup Q_{d+1}, d + 1$;
17: end while
```

Theorem 7. *Let Q_d be an arbitrary set of places obtained at iteration d in Algorithm 1. Let $V = \bigcup_{i<d} Q_i$ be the set of places that have been visited and $C = succ(Q_d) \setminus V$. If for all different places $s_1, s_2 \in Q_d$, $succ(s_1) \cap succ(s_2) \subseteq V$, then for all $s \in C$:*
(1) *if $|C| \leq N$ then $P(s \in Q_{d+1}) = 1$.*
(2) *if $|C| > N$ then $P(s \in Q_{d+1}) = \frac{N}{|C|}$.*

Proof. The first case is trivially satisfied, as in line 8 all non-revisited successor places are added to Q_{d+1} so long as the set Q_{d+1} is not full (i.e. $|Q_{d+1}| < N$). The second case is settled by means of an inductive proof. In particular, we prove that the addition of place t in line 11 should be done with probability $\frac{N}{c}$ and at random in the set. With induction on the difference $c - N$ we show that doing so leads to a uniform distribution (i.e. every place has probability $\frac{N}{|C|}$ of being in the set):
(1) Base case: $c - N = 0$. This is trivial.
(2) Inductive step: $c - N = k + 1$ for some k. We added the last place, t, with probability $\frac{N}{c}$, so we need only consider the other places. Before adding place t they are in the set with probability $\frac{N}{c-1}$ (by induction) and when the last place has been considered, they are still in the set afterwards with probability $\frac{N}{c}\left(1 - \frac{1}{N}\right) + \left(1 - \frac{N}{c}\right)$. That is, if the new element is added, then each currently selected place has chance $\frac{1}{N}$ of being removed (or $1 - \frac{1}{N}$ of not being removed). If t is not added, then the selected elements obviously remain selected. It is straightforward to derive that

$$\frac{N}{c-1}\left(\frac{N}{c}\left(1 - \frac{1}{N}\right) + \left(1 - \frac{N}{c}\right)\right) = \frac{N}{c}. \qquad \square$$

Indeed, Algorithm 1 correctly implements a highway simulation whenever two places in a lane only lead to the same place if that place was already visited; all other places that can be reached should be unique.

In practice, this property is difficult to assess upfront. In degenerate cases, the selection mechanism for the next places that can be reached by the current lanes in Algorithm 1 strongly favours *funnels*: eligible reachable places that are shared among a significant amount of places in the current lanes. This is confirmed by the following analysis.

Property 8. *Let Q be the selection at some point of some depth d of Algorithm 1 for width N. Further, let c be the number of places actually considered for addition up to that moment. We analyse the probability of a place to end up in a lane of the highway simulation when that place is considered for addition multiple times.*

Let $s \notin Q$ be a place. Suppose that we successively try to add s, K ($K > 0$) times. The probability that s is in Q afterwards is obtained as follows. Let probabilistic variable X_k denote the conditional probability of adding place s to Q after K attempts when s fails to be added for the first $k \leq K$ attempts

$$P(X_k) = \begin{cases} 0 & \text{if } k = K \\ \frac{N}{c+k+1} + (1 - \frac{N}{c+k+1})P(X_{k+1}) & \text{if } 0 \leq k < K \end{cases} \tag{1}$$

The above expression can be simplified, yielding the following probability function:

$$P(X_k) = 1 - \frac{(c+k)!(c+K-N)!}{(c+K)!(c+k-N)!} \tag{2}$$

For $k = 0$, we then obtain $P(X_0) = 1 - \frac{c!(c+K-N)!}{(c+K)!(c-N)!}$.

To illustrate the effect for an approximate highway simulation of width $N = 1$ we have that the probability that s will be selected is $\frac{K}{c+K}$ $\left(\text{as opposed to the more desirable } \frac{1}{c+1}\right)$.

We can see that K and N have a positive relation to the probability of duplicates being chosen and that c has a negative relation to it.

Next, observe that a place has the greatest probability of being selected if all of its occurrences are considered last. This implies that the probability that a place that occurs K times in a sequence of size M consisting of U $(U > N)$ unique elements is selected with a probability in the interval

$$\left[\frac{N}{M}, 1 - \frac{(U-1)!((U-1)+K-N)!}{((U-1)+K)!((U-1)-N)!}\right]. \tag{3}$$

From the above analysis, it follows that in degenerate cases, funnels are more likely to appear in the highway simulation than ordinary places.

Property 9. *When every place t occurs an equal amount of times as successor of a place $s \in Q_d$ and all successors are treated in a uniformly distributed way, then the probability of place t appearing in Q_{d+1} approaches $\frac{N}{U}$, where U is the number of unique successors.*

While the preceding analysis suggests a highway simulation and an approximate highway simulation behave very differently, this is not supported by the experiments that we conducted in Section 5. In these experiments, the difference turns out to be insignificant for most applications, suggesting that the degenerate cases really are less likely to occur than the optimal cases. We expect that this is due to the small probability of funnels to be present in small highway widths.

5. Experiments

We conducted a number of experiments on both classical and novel problems using three search techniques: random search, randomised depth-first search and approximate highway search. In the only case where approximate highway search differs significantly from the *ideal* highway search (see Section 5.2.3), we added the latter in our exposition of the particular experiment. In our experiments, we studied the efficacy of a search method to (1) find a state or event of interest and (2) find a short witness to that state or event of interest. Most results are summarised at the end of this section in Table 1; three cases, including the *dining philosophers* are discussed in greater detail as these illustrate different aspects of the search methods that are used. Note that we do not include explicit memory consumption data as for both the randomised depth-first search and the (approximate) highway search the memory consumption is linear in the number of visited states. For random search the memory usage is constant (apart from storing a trace in certain cases).

Apart from analysing the effectiveness of the search techniques on problems taken from practice, we first study the impact of commonly found substructures on the behaviour of the search techniques. The latter serves a better understanding of the intricacies of the search techniques.

All experiments have been conducted using the toolsuite mCRL2,[1] a process algebraic language with tool support for generating and analysing (infinite) state spaces. The systems that we used in our experiments can all be found in the mCRL2 distribution.

5.1. Complex (Sub)structures

Most transition systems stemming from real applications are of moderate overall structural complexity, but contain substructures that are particularly difficult to explore. Typical examples are *diamond structures*, that arise due to the interleaving semantics that is given to parallel components, and *back-loops* that appear due to recovery mechanisms or resets, *etc*.

5.1.1. Diamond structure

Diamond-structured state spaces are a known source of complexity for random-based search techniques [15]. The probability to enter into the outer ends of a diamond structure is virtually zero: the outermost states in a diamond of width n can only be reached via 1 of the 2^{n-1} paths of length $n-1$; in general,about 95% of the paths of length $n-1$ lead to just over 40% of the states at depth n.

Setup and analysis. Our experiments involve a diamond structure of width 10, shown in Fig. 2. The search objective is any state numbered 1–10 (indicated in Fig. 2 by the states 1 and 10 and the remaining 8 states at the widest part of the diamond structure). In our experiments, we analyse the probability of hitting one of the 10 numbered states in the state space by conducting 1000 experiments for each search method and noting the average probability of hitting a given numbered state. The resulting distributions of random search and highway searches of width 2, 4, 6, 8 and 10 are depicted in Fig. 2. A

[1] See http://www.mcrl2.org.

Table 1 Results from the various experiments; the value for succ reports the percentage of successful searches; the value reported for state indicates the average number of states required to find the searched state (measured only among the successful searches); the value reported for trace indicates the average length of the trace that witnesses the searched state (again measured only among the successful traces). The search objectives for the respective experiments were as follows: action 'LDres' (1394), action 'c10' (brp), action 'CFSAP_out' (commprot), action 'protocol_error' (lift-init), action 'Draw' (othello), and action 'error' (swp).

	Random			rdfs			Highway 8			Highway 16			Highway 32		
	States	Trace	Succ.	States	Trace	Succ.	States	Trace	Succ.	States	Trace	Succ.	States	Trace	Succ.
1394(2)	3266	158	100	981	491	100	1154	174	88	2078	153	93	3439	125	95
1394(4)	4870	3777	100	2990	2501	99	1807	322	86	2047	175	99	3514	141	99
brp(10)	268	137	100	155	152	100	937	124	100	1441	94	100	2390	78	100
brp(30)	19885	311	75	382	361	100	1717	240	100	3089	209	100	5611	186	100
brp(50)	0	0	0	539	494	100	2345	341	100	4496	315	100	8415	286	100
brp(70)	0	0	0	690	629	100	3043	445	100	5809	413	100	11372	387	100
brp(90)	0	0	0	821	745	100	3593	542	100	7103	509	100	14094	486	100
commprot	1516	1517	100	4864	631	72	7418	929	100	13,181	825	97	16,441	515	90
lift-init(3)	13	14	2	553	14	100	87	13	31	149	13	58	257	13	93
lift-init(4)	0	0	0	1232	17	100	106	15	14	207	16	45	367	15	81
lift-init(5)	0	0	0	2438	19	100	119	17	2	253	18	30	461	17	65
lift-init(6)	0	0	0	6311	21	99	0	0	0	313	22	11	542	20	48
othello(4×4)	12	13	12	86	13	26	87	13	49	165	13	77	312	13	91
othello(6×6)	32	33	3	300	33	20	248	33	35	487	33	58	955	33	80
swp(2)	327	204	27	279	274	27	1217	154	26	2214	141	27	3161	102	27
swp(4)	1640	906	27	1570	1552	27	6307	790	27	12,794	802	27	13,374	421	27
swp(6)	4391	2372	27	3872	3844	27	14,986	1875	27	24,588	1540	27	29,377	922	27
swp(8)	7273	3315	27	8531	6500	25	21,419	2680	27	35,119	2198	26	48,052	1505	21
swp(10)	11,546	5642	27	16,327	8115	21	29,425	3681	26	45,012	2816	22	62,502	1957	12

randomised depth-first search yields a consistent 100% for all values as it explores all 100 states; this is due to the finiteness of the state space. Likewise, a highway search of width 10 finds all numbered states, but compared to a randomised depth-first search, the latter requires only half as many states to be explored.

The effect of the probabilistic approaches used in a random search and in highway searches of small width is clearly visible in Fig. 2. As expected, a random search covers fewer states than a highway search and highway searches of increasing width cover more states more often.

5.1.2. Back-loops

Back-loops occur frequently due to *system resets* and *recovery mechanisms*. A random search tends to revisit states instead of exploring new states in such situations.

Setup and analysis. We used a back-loop system consisting of 47 states (see Fig. 3) to demonstrate the efficacy of all search techniques in finding the (single) deadlock in the system. The plots in Fig. 3 show the probability of each search method of finding the deadlock against the number of states that have been visited.

Note that randomised depth-first search performs quite good; only a highway search of width 5 performs better after 30 states have been investigated. The results of random search confirm its revisiting behaviour; even after 50 steps it cannot find the deadlock with a significant probability. A highway search of width 1 (not shown) is outperformed by all searches due to the inability to revisit states. The results of highway searches show that each width has an optimum that can be achieved; exploring more states does not lead to better detection rates, which again can be explained by the inability to revisit states.

5.1.3. Strongly connected components

Substructures that contain a *strongly connected component* "absorb" a probability-based search: the search is confined to a single SCC. The behaviour of a highway search is, due to a more balanced search strategy that allows it to investigate multiple SCCs, subtly different.

Setup and analysis. The state space we study consists of five reachable strongly connected components with single exits, see Fig. 4. The impact of the number of states to be explored on each search method's capability of finding one of the numbered states is also depicted in Fig. 4.

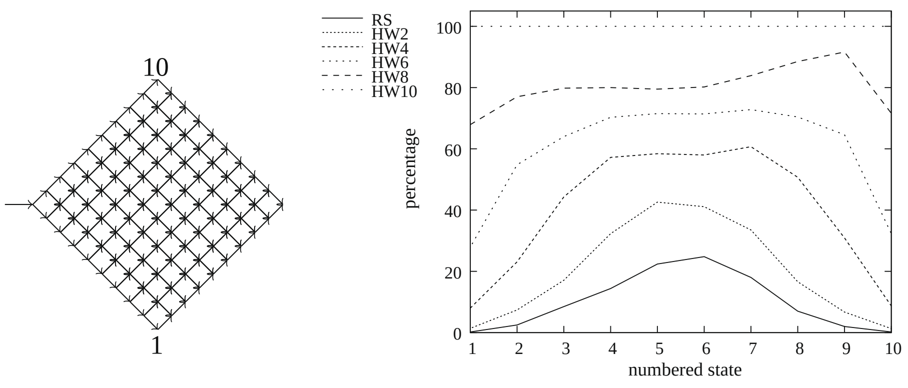


Fig. 2. Diamond structure of width 10 (left) and search results (right).

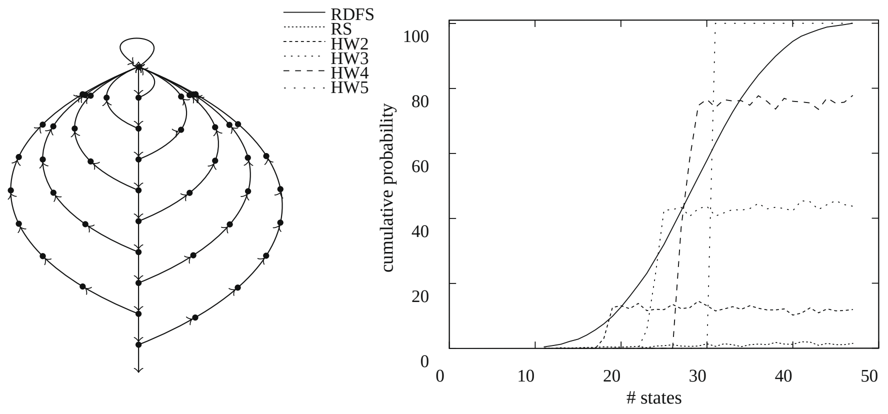


Fig. 3. Back-loop system of 47 states (left) and search results (right).

Randomised depth-first search, being an exhaustive search method, examines each connected component only once (but exhaustively) before turning its attention to another component. This causes the stepwise behaviour as shown in Fig. 4. A highway search of width 1 slightly outperforms a random search as it will always find (precisely) one exit while random search sometimes misses one because of revisits. At wider widths, a highway search starts to approach a randomised depth-first search.

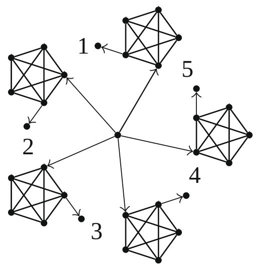
5.2. Applications

The results of previous section do not straightforwardly transfer to practical situations, although they do provide additional intuition behind each search method's capabilities. In this section, we study the effectiveness of the search techniques on documented cases which are either inherently large or can be scaled up to become too large for model checking purposes. We study the differences in typical coverage of the state spaces using all three search methods. Three systems are studied and described in greater detail, as these clearly illustrate the differences in the search methods. Table 1 summarises the results of a large number of experiments; in particular, it lists the percentage of successful searches, and for the successful searches, it provides the average number of states and the average length of a witness for finding states of interest.

5.2.1. A network of buffers

The analysis of system performance is computationally quite costly or even intractable. As an example, we study the performance analysis of a network of buffers. Variabilities such as buffer capacity, channel speeds, mutual exclusion protocols, *etc.* affect throughput in ways that are hard to predict.

Setup and analysis. The setup depicted in Fig. 5 is a simplification of a proprietary industrial protocol [4] for quickly collecting data from a large number of distributed receivers; each buffer and channel in the network introduces its own delay, and a mutual exclusion protocol is used to enforce singular communications between the buffer and the receiver (the *Merge* in Fig. 5). Data arrives at the buffers and is propagated to the receiver as soon as the latter is capable of processing the data. We search for extreme propagation times of 100 data units through the network configuration of Fig. 5.



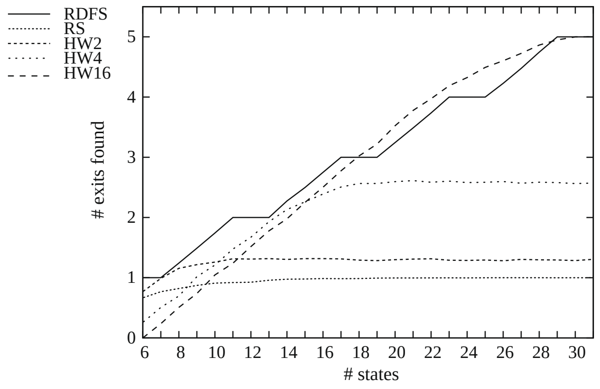


Fig. 4. Five strongly connected components (left) and search results (right).

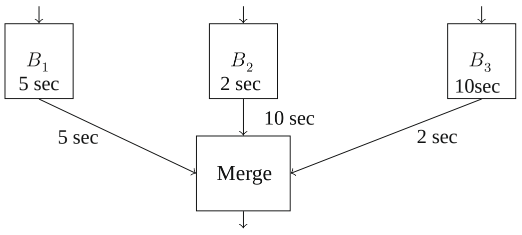


Fig. 5. A network configuration of depth 2 and width 3, with propagation times and channel delays in seconds.

Our model of the network protocol terminates after 100 data units have been received by process *Merge*. Since the protocol does not contain loops or cycles, each path in the protocol is finite and equally long (202 states). We set an upper bound of $202 \times N$ on the number of states that are allowed to be explored by each search method (N depends on a highway search width).

In Fig. 6 we report the minimal and maximal propagation times that are found by runs of each search method (limited to $202 \times N$ states). Comparing the intervals of propagation times that are reported by each search method, we find that a highway search of width 2 is comparable to a random search, and outperformed by a randomised depth-first search. For highway searches of width 1000 and 2000, we see that highway search ranks highest, followed by a randomised depth-first search and finally random search. The difference in performance between wide highway searches (widths 1000 and 2000) on one end and randomised depth-first search on the other end is explained from the fact that a highway search acts as a restricted breadth-first search, i.e. it covers a wide portion of the state space in a homogeneous fashion, whereas a randomised depth-first search starts searching the behaviour of the network protocol given a single scenario, and performs local back-tracing to find subtle variations on it. The poor coverage of a random search can be explained from the fact that the structure of the state space of the network protocol has characteristics of a diamond structure.

Note that a single run is not necessarily representative for the extremities that can be found using random search or highway search. Fig. 6 shows the number of occurrences of reported propagation times for 100 highway searches of width 1000 compared to a 100,000 random searches. It shows that a random search is less likely to find extreme values than a highway search and that most values that are found by a random search are centred around a propagation time of 1100. The graph of Fig. 6 does not contain a picture of a randomised depth-first search as the number of propagation times it finds is several magnitudes larger; this is due to the fact that the finiteness of the protocol enables short back-trackings of the randomised depth-first search, thereby reporting an extremely large number of (similar) propagation times.

5.2.2. Automated parking garage

In [14] an architecture for an automated parking garage is developed and verified. The main obstacles in verification of the design is its very large state space. Instead of applying abstraction to reduce the state space to manageable proportions (as was done in [14]), we analyse the original specification, which contains two known faults. A complicating factor in this design is that the search space contains many loops and cycles.

Setup and analysis. We are interested in the (cumulative) probability of finding the error upon having examined a given number of states, and a given simulation length, respectively.

method → ↓ N (width)	RS	RDFS	HS$_N$
1	1129	1056	1124
2	1065 – 1075	1043 – 1194	1142 – 1147
1000	1014 – 1194	1010 – 1246	1010 – 1265
2000	1016 – 1196	1017 – 1232	998 – 1284

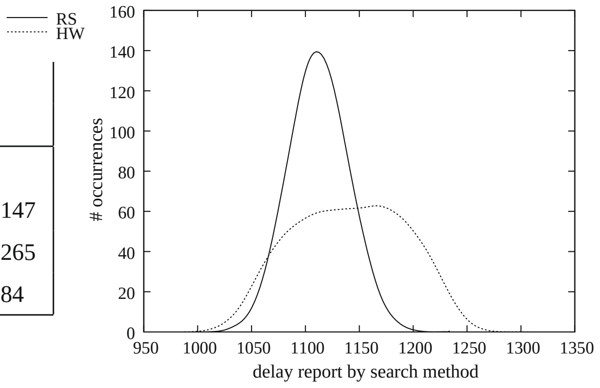


Fig. 6. Minimal and maximal propagation times found using random search, randomised depth-first search and highway search of widths 1, 2, 1000 and 2000 (left), and typical number of propagation times found by a random search and highway Search of width 1000.

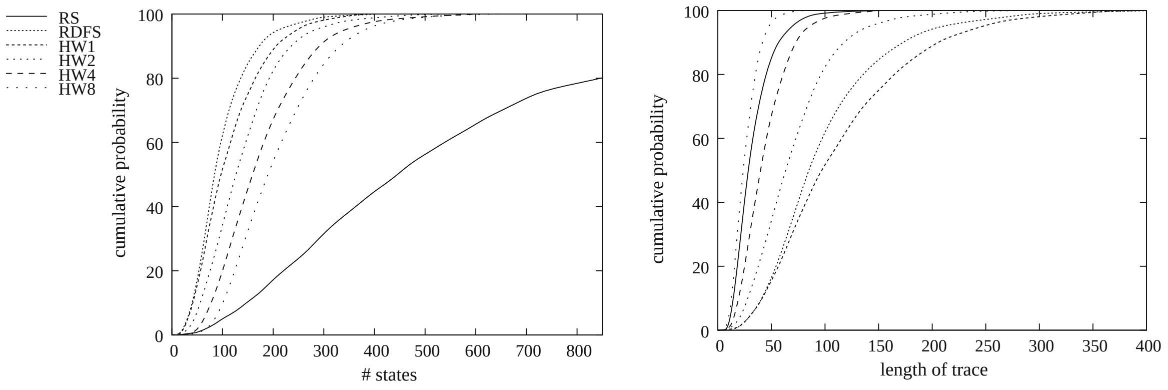


Fig. 7. Search results for the automated parking garage.

The results are depicted in Fig. 7. On average, a random simulation visiting 4000 states is needed to find the error. In contrast, a highway search of width 1 outperforms random search significantly, requiring a simulation that visits less than 400 states to reach the error. A randomised depth-first search is most successful at finding the error quickly.

When we look at the simulation lengths that are required to find an error, a different picture emerges: we find that a highway search of width 8 leads to shortest witnesses, followed by a random search. Using this metric, a randomised depth-first search performs rather poorly. The difference can be explained from the breadth-first search-like characteristics of highway search. The rather successful results of random search are likely to be linked to observations of [15], stating that increasing the time spent in a random search has only a small effect on the number of new states that are visited; in this sense, finding the error at some point means that it is relatively close to the initial state.

5.2.3. Dining philosophers

The *dining philosophers* problem is a problem that clearly illustrates the subtleties and complexity of synchronisation in concurrency. An advantage of the problem is that it is highly scalable, yet always exhibits a single deadlocking situation. We have examined two instances of the dining philosophers problem: an instance with 5 and an instance with 17 philosophers.

Setup and analysis. The deadlock search problem in the instance of the dining philosophers problem with 17 philosophers is inspired by [5]. The results of a highway search of width 2–5 and a random search and a randomised depth-first search are depicted in Fig. 8. Each search is terminated after exploring 5000 states, which explains why not all searches yield a 100% chance of finding the deadlock.

Surprisingly, random search outperforms all other search techniques, and only 70% of the randomised depth-first searches find the deadlock. Even more surprising is the observation that highway searches of greater width perform poorer than highway searches of smaller width, meaning that the witnesses leading to the deadlock states become more complex at increasing highway search widths.

We use the instance of five philosophers to further illustrate these findings. Apart from providing our findings for the approximate highway search, we also provide our findings for the ideal highway search; this demonstrates that the observed

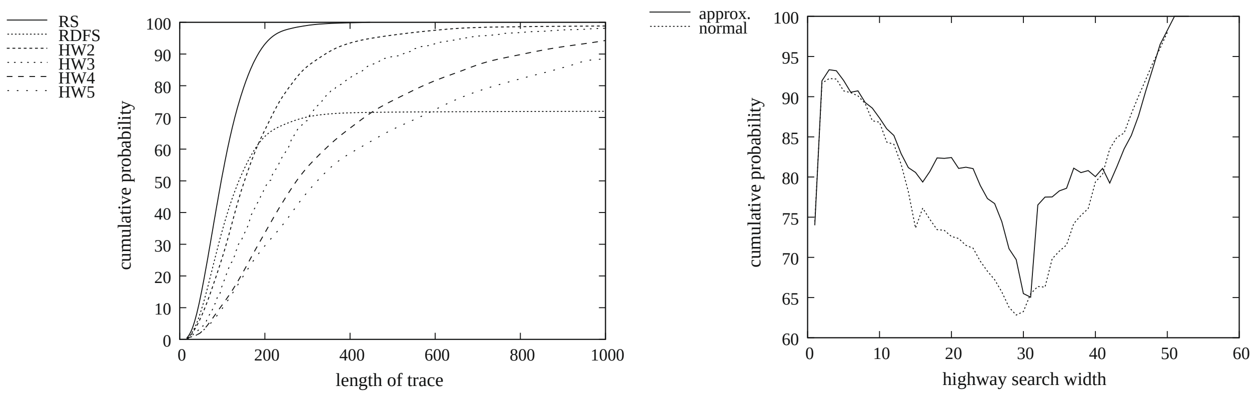


Fig. 8. Cumulative probability of finding the deadlock in the dining philosophers (instance 17) using various search methods (left), and total probability of finding the deadlock in the dining philosopher's problem (instance 5) at increasing highway widths and approximate highway widths.

phenomenon is not an artefact of the approximation algorithm. In Fig. 8, the width of a highway search (horizontal axis) is plotted against the total probability of detecting the deadlock. The probability is obtained as an average of 4000 experiments for each highway search width. Note that each search terminates, regardless of whether the deadlock is found or not. This is due to the inevitable revisits of states, as the total state space is finite. Remark that for this experiment, we did not limit the number of states that are allowed to be explored.

An explanation for this remarkable situation can be found in the unusual structure of the state space which has features of the *back-loops* problem of the previous section. Increasing the width of a highway search increases the probability that paths to the deadlocking state get disabled. Since one can only return to a path to a deadlock if one revisits a state earlier on this path (which is impossible in a highway search), the total probability of reaching a deadlock decreases.

Note that Godefroid and Kurshid [5] also studied the dining philosophers problem with 17 philosophers; they compared the efficacy of their genetic search algorithm against the performance of random search and concluded that their search strategy was superior. In their experiments, they limited the runs of the random searches to explore 68 states before resetting the search. The authors report finding no deadlock in 8 hours run-time using random search. Unfortunately, we cannot confirm these findings using our tooling; on the contrary: we find the deadlock in 20% of the cases within seconds when using random searches of at most 68 states (see Fig. 8). It is unclear to us what causes these differences in observations. A separate analysis conducted on our side confirms that our random search is as random as can possibly be achieved.

5.2.4. Benchmarks summary

The experiments conducted on the complex subsystems, and the previous three experiments suggest that in many instances, a randomised depth-first search requires the least amount of states to find a state of interest in a given state space. Highway search, on the other hand, often yields the shortest witnesses to states of interest. Table 1 substantiates these findings: highway search consistently outperforms randomised depth-first search when it comes to finding short witnesses, even at relatively small widths; on the other hand, a randomised depth-first search yields faster results. It is noteworthy that random search also yields shorter witnesses compared to randomised depth-first search for 8 out of the 19 experiments and performs equally well on another 3 experiments.

The experiments consist of instances of the bounded retransmission protocol [8] in which the timeout setting is varied; Tanenbaum's original faulty sliding window protocol [19] (see [2] for a formal specification; the fault is a nasty livelock) with varying window sizes; the IEEE 1394-Firewire protocol [13] with a varying number of link layers; the conference protocol (commprot) [1]; a distributed truck lift system [7] with varying number of lifts and the game Othello with a varying board size. For all searches except for sliding window protocol, we limited the search to 50,000 states; the searches of the Sliding Window Protocol are aborted at 80,000 states. For each problem instance, Table 1 notes the value specific parameter that is used in between brackets.

6. Conclusion

We studied a novel search technique, called *highway search*. The search technique can be used to find states of interest in large and infinite state spaces, and has the advantage over more dedicated techniques such as *directed model checking* techniques that it does not require expert user knowledge. We have compared the efficacy of highway search with two other push-button search techniques, viz. *depth-first search* and *random search*. A small study of the search techniques on small yet commonly occurring substructures is conducted to obtain a better insight into the intricacies of the three search techniques.

A further comparison of all three search methods is done by means of an evaluation of the search techniques on large state spaces that come from practice or appeared in the literature, see Section 5.2. Summarising the results, we find that randomised depth-first search is most efficient at finding states of interest fast (i.e., using the least number of states that are visited), while a highway search clearly is the most appropriate tool for finding short witnesses that lead to the states of interest.

The observed efficacy at finding interesting states *fast* of the randomised depth-first search method confirms the findings of [17], where it is furthermore suggested to use a randomised depth-first search as a yardstick for measuring the efficacy of directed model-checking approaches. However, as we argue, efficiency is only a part of the goals in validation; explanation of the findings is equally important, if not more so. Taking lengths of witnessing traces as a yardstick, highway search is the more appropriate tool for comparing directed model checking techniques against. We remark that increasing the highway search width leads to shorter witnesses explaining the reachability of interesting state, at the cost of more states that need to be explored. The limit case, where highway search and breadth-first search coincide is in many cases not of practical interest due to the large amount of memory required by a breadth-first search.

In order to reduce memory requirements and optimise time efficiency, we presented an alternative algorithm approximating a highway simulation (which underlies a highway search). We proved that under specific conditions, it faithfully implements a highway simulation, and that its worst-case behaviour is significantly different from the ideal highway simulation. In practice, this difference is hardly ever experienced, and, in the only case in which we found a significant difference, we found that approximate highway search outperforms ideal highway search.

Appendix

A. Specifications of Complex (Sub)structures

In order to be able to repeat the experiments conducted in previous sections, we list all three complex substructure specifications given in mCRL2 of Section 5.1. The specifications that have been used in the experiments on practical applications can be found online in the repositories of mCRL2.

Diamond structure

```
% A diamond structure of N+1 states wide;
% total size of the state space is (N+1)^2 states


map N : Int; eqn N = 9;

act report : Int;


proc X(i,j: Int) =
 (i+j < N) -> (tau. X(i+1,j) + tau. X(i,j+1)) <> delta
+
 (i+j >= N && i + j < 2*N+1) ->
   (
    (i + j == N ) -> report (j). X(i,j)
  +
    (i +j >= N && j < N) -> tau. X(i, j+1)
  +
    (i +j >= N && i < N) -> tau. X(i+1,j)
   ) <> delta;

init X(0,0);
```

Back-loops

```
% A system with a single trace to the end with backpointers to
% the root at all nodes in between.


map N : Int;
eqn N = 9;    % N+1 is the deadlocking node
```

```
act report:Int;

proc P(n : Int) =
  (n < N) -> tau. P(n+1)
+ (n == N) -> tau. delta + (n > 0 && n <= N) -> Q(n-1,n-1) + (n ==
0) -> tau. P(n);


proc Q(n: Int, m: Int) =
  (n == 0) -> tau. P(0)
+ (n >= 1) -> tau. Q(n -1,m);


init P(0);
```

Strongly connected components

```
% A system with an initial state and scc_num (=5) SCCs
% each SCC consists of scc_size (=5) states.

map scc_num, scc_size: Pos;
eqn scc_num = 5;
    scc_size = 5;

act ini, scc, report: Pos;

proc P = sum id: Pos. (id <= scc_num) ->
                        ini(id).SCC(id,1,scc_size);

     SCC(id: Pos, pos: Pos, size: Pos) =
         sum new_pos: Pos.
           (new_pos <= size && new_pos != pos) ->
             scc(id).SCC(id,new_pos,size)
       + (pos == size) -> report(id).D(id);

     D(id: Pos) = delta;

init P;
```

References

[1] A. Belinfante, J. Feenstra, R. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, L. Heerink, Formal test automation: a simple experiment, in: G. Csopaki, S. Dibuz, K. Tarnay (Eds.), Twelfth Int. Workshop on Testing of Communicating Systems, Kluwer Academic Publishers, 1999

[2] J. Brunekreef, A formal specification of three sliding window protocols, Tech. Rep. P9102, CWI (1991).

[3] M. Dwyer, S. Elbaum, S. Person, R. Purandare, Parallel randomized state-space search, in: Proceedings of ICSE'07, IEEE Computer Society, 2007.

[4] T. Engels, Analysis of the iRF Digital Network, Master's thesis, Eindhoven University of Technology (2007).

[5] P. Godefroid, S. Khurshid, Exploring very large state spaces using genetic algorithms, in: J.P. Katoen, P. Stevens (Eds.), Proceedings of TACAS 2002 of, LNCS, Vol. 2280, Springer-Verlag, 2002.

[6] A. Groce, W. Visser, Heuristics for model checking java programs, Software Tools for Technology Transfer 6 (4) (2004) 260–276.

[7] J. Groote, J. Pang, A. Wouters, Analysis of a distributed system for lifting trucks, Journal of Logic and Algebraic Programming. 55 (1–2) (2003) 21–56.

[8] J. Groote, J. van de Pol, A bounded retransmission protocol for large data packets, in: M. Wirsing, M. Nivat (Eds.), Proceedings of Algebraic Methodology and Software Technology, 5th International Conference, AMAST '96, vol. 1101 of Lecture Notes in Computer Science, Springer, 1996.

[9] G. Holzmann, The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley, 2004.

[10] M. Jones, J. Sorber, Parallel search for LTL violations, Software Tools for Technology Transfer 7 (2005) 31–42.

[11] S. Kupferschmid, K. Dräger, J. Hoffmann, B. Finkbeiner, H. Dierks, A. Podelski, G. Behrmann, Uppaal/DMC – Abstraction-based Heuristics for Directed Model Checking, in: O. Grumberg, M. Huth (Eds.), Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems of, Lecture Notes in Computer Science, Vol. 4424, Springer-Verlag, Berlin Heidelberg, 2007.

[12] S. Kupferschmid, M. Wehrle, B. Nebel, A. Podelski, Faster than uppaal?, in: Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008), vol. 5123 of Lecture Notes in Computer Science, Springer-Verlag, 2008.

[13] S. Luttik, Description and formal specification of the Link Layer of p1394, Tech. Rep. P9706, CWI (1997).

[14] A. Mathijssen, A. Pretorius, Verified design of an automated parking garage, in: L. Brim, B. Haverkort, M. Leucker, J. van de Pol (Eds.), Formal Methods: Applications and Technology, of Lecture Notes in Computer Science, Vol. 4346, Springer, 2007.

[15] R. Pelánek, T. Hanžl, I. Černá, L. Brim, Enhancing Random Walk State Space Exploration, Tenth International Workshop on Formal Methods for Industrial Critical Systems (FMICS 05), ACM Press, New York, NY, USA, 2005.
[16] J. Rasmussen, G. Behrmann, K. Larsen, Complexity in simplicity: Flexible agent-based state space exploration, in: O. Grumberg, M. Huth (Eds.), Proceedings of TACAS of Lecture Notes in Computer Science, Vol. 4424, Springer-Verlag, 2007.
[17] N. Rungta, E. Mercer, Hardness for explicit state software model checking benchmarks, in: Proceedings of SEFM 2007, IEEE Computer Society, 2007.
[18] H. Sivaraj, G. Gopalakrishnan, Random walk based heuristic algorithms for distributed memory checking, in: Proceedings of Parallel and Distributed Model Checking (PDMC'03), vol. 89(1) of ENTCS, Elsevier, 2003.
[19] A. Tanenbaum, Computer Networks, Prentice Hall, 1988.
[20] W. Visser, K. Havelund, G. Brat, S. Park, Model checking programs, in: Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE), IEEE Computer Society, 2000.
[21] A. Wijs, B. Lisser, Distributed extended beam search for quantitative model checking, Proc. 4th Workshop on Model Checking and Artificial Intelligence (MoChArt'06), of LNAI, Vol. 4428, Springer-Verlag, 2007.