A New Scheme for Memory-Efficient Probabilistic Verification

Ulrich Stern and David L. Dill
Stanford University
Department of Computer Science, Stanford University, Stanford, CA 94305, USA
Email: {uli@beet,dill@cs}.stanford.edu

Abstract

In verification by explicit state enumeration, for each reachable state of the protocol being verified the full state descriptor is stored in a state table. Two probabilistic methods – bitstate hashing and hash compaction – have been proposed in the literature that store much fewer bits for each state but come at the price of some probability that not all reachable states will be explored during the search, and that the verifier may thus produce false positives. Holzmann introduced bitstate hashing and derived an approximation formula for the average probability that a particular state is not omitted during the search, but this formula does not give a bound on the probability of false positives. In contrast, the analysis for hash compaction, introduced by Wolper and Leroy and improved upon by Stern and Dill, yielded a bound on the probability that not even one state is omitted during the search, thus providing a bound on the probability of false positives.

In this paper, we propose and analyze a new scheme for probabilistic verification that is a variation of the improved hash compaction scheme. The main difference is that a tighter bound on the probability of false positives is calculated by reasoning about a longest path in the breadth-first search tree of the reachable state space. In addition, the new scheme employs ordered hashing to reduce the omission probability when inserting into the state table. In the industrial protocols we examined, the new scheme yielded an exponential reduction in the bound on the probability of false positives, which enabled a roughly 50% reduction in the number of bits needed for a compressed state. Furthermore, we propose a memory efficient way to store the information needed for error trace generation. The outcomes of experiments using the new scheme confirmed the analysis.

Keywords

Probabilistic verification, explicit state enumeration, reachability analysis, bitstate hashing, hash compaction, Murφ verifier

1 INTRODUCTION

Complex protocols are often verified by examining all reachable protocol states from a set of possible start states. This reachability analysis can be done using two different methods: the states can be explicitly enumerated by storing them individually in a table, or a symbolic method can be used, such as representing the reachable state space with a binary decision diagram (BDD) (Burch et al. 1990).

The biggest obstacle of both methods is the often unmanageably large number of reachable states – the 'state explosion problem'. Symbolic methods can alleviate the state explosion problem in some cases. However, in research done in our group for some types of industrial protocols, explicit state enumeration has out-performed the symbolic approach (Hu et al. 1994).

In explicit state enumeration algorithms, a state table is maintained that eventually holds all reachable states of the protocol under verification unless an error is detected. This state table is usually implemented as a hash table. In practice, the total memory available for the hash table is the limiting resource in verification. Therefore, it is desirable to use the most compact representation for this table.

When using a conventional table, for each state the full state descriptor is stored, which typically has several hundreds of bits. Two probabilistic methods – bitstate hashing and hash compaction – have been proposed that store many fewer bits for each state but come at the price of some probability that not all reachable states will be explored during the search, and that thus protocol errors might go undetected. In the case that all protocol errors are undetected the verifier is said to produce **false positives**.

Bitstate hashing, introduced by Holzmann (1987, 1991), maintains a table of bits which are initially set to zero instead of maintaining a conventional state table. When a state is inserted into the table, two hash values are calculated from the state descriptor and the bits corresponding to these hash values are set to one. When the algorithm examines a newly reached state and finds that the two bits corresponding to this state are already set to one, it assumes that it has visited the newly reached state earlier. Holzmann (1995) also presented an approximate analysis for the average probability that a particular state is not omitted. Unfortunately, this **average** does not provide us with a **bound** on the omission probability for a particular state. Some states might have much higher omission probabilities than this average. Intuitively, these are the states which are only reachable by one or few long paths, since these states can only be visited if many other states are visited as well. Thus, errors might go undetected with high probability and we have no bound on the probability of false positives, even if the average probability that a particular state is omitted is small. Hence, when a verifier based on this scheme reports 'no errors', it is impossible for the user to assess the probability (bound) that this report is false.

In contrast, Wolper and Leroy (1993) presented an analysis for a new scheme, which they called hash compaction, that yields an upper bound on the probability that the search becomes incomplete, i.e. that one has even one omission during the search. To achieve this improvement in the reliability of the scheme, the hash compaction method requires more memory: it stores compressed values with typically 64 bits for each state in a conventional hash table. Furthermore, Wolper and Leroy found the hash compaction scheme to be slightly superior to a bitstate hashing scheme in which multiple instead of two bits/hash values are used for each state. When a verifier based on hash compaction

reports 'no errors', it can also report a bound (typically 0.1%) on the probability that the report is false.

An improvement of the hash compaction scheme was introduced by Stern and Dill (1995b), which differs from the previous scheme in two ways. First, the improved scheme uses open addressing instead of chaining to resolve collisions in the hash table. Chaining requires storing an additional pointer besides the compressed state. Since a pointer has roughly the same size as the compressed state, chaining roughly doubles the memory requirements. Second, in the improved scheme the probe sequence of a state – the indices for searching the hash table – is calculated independently from the compressed value. Thus, the number of bits one has to store for each state is reduced to typically 40. In the improved scheme, the probability that a particular state is omitted during its insertion into the hash table is roughly proportional to the length of the probe sequence used during the insertion.

In this paper, we present and analyze a new probabilistic verification scheme that is a variation of the improved hash compaction method. The major difference is that the new algorithm calculates a bound on the omission probability for each **particular** state by reasoning about a longest path in the breadth-first tree of the reachable state space. The new analysis yields a tighter bound on the probability of false positives and thus requires fewer bits per state for the same value of the bound on the probability. The analysis is based on the insight that an error state will not be omitted if no state is omitted on some **path** from the startstate to the error state. The omission probability for the error state thus depends on the length of the paths leading to it. By employing breadth-first search, the analysis can use a shortest path to the error state for calculating a bound on the omission probability. Intuitively, if one has a large state space with one error state, the error state will be explored if none of possibly many omissions hits the shortest path to the error state. The previous analysis, in contrast, allowed not even one omission among the reachable states. Furthermore, the new scheme employs ordered hashing (Amble and Knuth 1974), which can reduce the length of the probe sequences for unsuccessful searches in open addressing and thus also reduce the omission probability.

If n denotes the number of states in the reachability graph and d is its diameter, the new scheme roughly saves a factor of n/d in the bound on the probability of false positives in comparison to the previous hash compaction scheme, or roughly $\log_2(n/d)$ bits in the compressed state descriptor. In the industrial protocols we examined, the ratio of n/d grew exponentially with the number of processors of the protocols, which enabled a roughly 50% reduction in the size of the compressed values.

We implemented the new scheme in the context of the Murφ verification system developed at Stanford (Dill et al. 1992). The new implementation includes a memory efficient method to store the information needed for error trace generation. A file is used to store this information instead of the main memory. In experiments with one artificial and two industrial protocols, we measured the probabilities of false positives by running the verifier 100 times and compared the resulting values with the bounds on this probability obtained by the analysis. In all three experiments, the bounds on the probability yielded by the analysis were conservative but still quite accurate. This accuracy is remarkable, since the analysis uses only one of the shortest paths to each state, while typical protocols usually have many paths leading to a state, which, intuitively, might reduce the probability of false positives.

The paper is organized as follows. Section 2 reviews the known probabilistic methods

for explicit state enumeration. The new scheme is presented and analyzed in Section 3. In Sections 4 and 5, we discuss the implementation of the new scheme and describe the results on sample protocols. Finally, Section 6 contains some conclusions and suggestions for future research.

2 PROBABILISTIC METHODS FOR EXPLICIT STATE ENUMERATION

2.1 Explicit state enumeration

In explicit state enumeration, the automatic verifier tries to examine all reachable states from a set of possible start states. Either breadth-first or depth-first search can be employed for the state enumeration process. Both the breadth-first and the depth-first algorithms are straightforward. A depth-first algorithm was, for example, given by Wolper and Leroy (1993).

Two data structures are needed for performing the state enumeration. First, a state table stores all the states that have been examined so far and is used to decide whether a newly-reached state is old (has been visited before) or new (has not been visited before). Besides the state table, a state queue holds all active states (states that still need to be explored). Depending on the organization of this queue, the verifier does a breadth-first or a depth-first search.

The state table will eventually hold all reachable states, unless the number of states exceeds the capacity of the table, in which case the verifier halts with an error message. In practice, the total memory available for the table is the limiting resource in verification.

2.2 The bitstate hashing method

The bitstate hashing or supertrace method introduced by Holzmann (1987, 1991) aims at reducing the memory requirements for the state table. Instead of maintaining a conventional table in which the state descriptors are stored, bitstate hashing only maintains a huge table of bits which are initially set to zero. When a state is inserted into the table, two hash values are calculated from the state descriptor and the bits corresponding to these hash values are set to one. When the algorithm examines a newly reached state and finds that the two bits corresponding to this state are already set to one, it assumes that it has visited the newly reached state earlier. If the assumption is incorrect, this results in the **omission** of the newly reached state and – since the search algorithm backtracks when it finds a state that has been searched already – may potentially cause the omission of all successor states of the omitted state.

Holzmann (1995) also derived approximation formulas for the expected **coverage** of the bitstate hashing method, where coverage is defined as the percentage of the reachable states actually explored by the verifier. Note that the expected coverage equals the average probability that a particular state is not omitted, where the average is calculated over all reachable states. However, this **average** does not provide us with a **bound** on the omission probability for a particular state. Some states might have an omission probability which is much higher than this average. Intuitively, these are the states which are only reachable

by one or few long paths, since these states can only be visited if many other states are visited as well.

Clearly, the states with a high omission probability might very well be the only states to display protocol errors, whose exposure is the goal of verification. Therefore, the bitstate hashing method does not provide a bound on the probability that the verifier misses the protocol errors and produces false positives. In addition, the omission probability of these states might be close to one, so that even re-running the verifier with independent hash functions will not significantly reduce the high omission probability.

2.3 The hash compaction method

Basic scheme

The hash compaction method was introduced by Wolper and Leroy (1993). Like bitstate hashing, hash compaction aims at reducing the memory requirements for the state table. However, hash compaction stores a compressed state descriptor in a conventional table instead of setting two bits corresponding to hash values of the state descriptor in a table of bits.

Furthermore, Wolper and Leroy derived an upper bound on the probability of omitting **even one** state, which is the probability that full coverage is not achieved. Note that this probability is different from the probability that a particular state is omitted, whose average over all states was approximated by Holzmann for bitstate hashing. We will call the probability that a particular state is omitted **state omission probability** and the probability of omitting even one state **omission probability**.

In hash compaction, a compression function c is used to obtain the compressed value $c(s) \in \{0, \ldots, l-1\}$ to be stored in the table for each state s. Here, l denotes the number of all possible compressed values. If we use b bits for these values, $l = 2^b$.

In practice, one can use universal hashing (Carter and Wegman 1979) to calculate the compressed value from the state descriptor, as suggested by Wolper and Leroy (1995) . Then, the probability that two different states have the same compressed value is bounded, namely $\Pr(c(s_1) = c(s_2)) \leq 1/l$, for all $s_1, s_2 \in S$, $s_1 \neq s_2$, where S denotes the set of all possible states. This bound is achieved by choosing the compression function at random from a universal class of hash functions at the start of the verifier, thereby defeating possible adversaries. A consequence of this bound is that the omission probability will also be bounded.

Improvements to the basic scheme

Typically, the state table is implemented as a hash table. One problem in hashing is the occurrence of collisions. A collision occurs if two states hash to the same slot in the table. This collision can be resolved by either chaining or open addressing (Cormen et al. 1990). Chaining requires storing an additional pointer besides the compressed state. However, a pointer has roughly the same size as the compressed state and thus chaining approximately doubles the memory requirements. Consequently, open addressing is preferable over chaining for hash compaction.

In open addressing, a vectorial hash function $\underline{h}$ is applied to each state s yielding a probe sequence $h_0(s), h_1(s), \ldots, h_{m-1}(s)$, where m denotes the number of slots in the hash table and $h_i(s)$ is an index into the table. When inserting a state in the table, the slots are

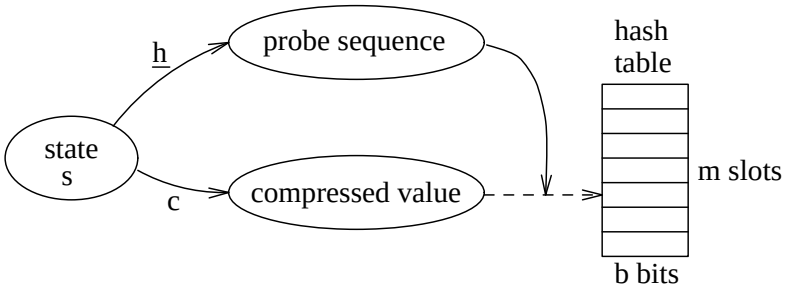

Figure 1 The state insertion process in the hash compaction scheme

tested for emptiness according to this probe sequence. The compressed state $c(s)$ is stored in the first empty slot found during the probe sequence. Note, that each probe sequence has to be a permutation of $\{0, \ldots, m-1\}$ if we want every slot in the table to be used.

The state insertion process is depicted in Figure 1. A probe can yield three different results:

1. The probed slot may be empty. The state s has not been encountered previously in the search and its compressed value $c(s)$ is stored in the slot.
2. The probed slot contains a compressed state different from the compressed state $c(s)$ being entered. This is called a **collision**. The insertion algorithm then probes the next slot given by the probe sequence $\underline{h}(s)$.
3. The probed slot contains a compressed state equal to the compressed state $c(s)$ being entered. In this case, the algorithm assumes that the uncompressed states are the same, which may or may not be true. The state table is not modified, and the successors of the current state s are not generated and searched. When the uncompressed states are indeed equal, this is the desired result. When the uncompressed states are not equal, this results in an **omission** of the current state s, and the possible omission of its successors in the state graph.

Observe that only the slots examined using the probe sequence can lead to omissions. Usually, only a few slots are examined before an empty slot is found (which is why hashing is attractive in the first place). Thus, if one calculates probe sequence and compressed value independently, the same compressed value can be stored at several locations without leading to omissions.

The improved hash compaction scheme – using open addressing and an independent calculation of probe sequence and compressed value – was presented and analyzed by Stern and Dill (1995b). With the improved scheme, it is possible, for example, to store 80 million states in a hash table of size 400 megabytes with an omission probability smaller than 0.13% using 40-bit compression.

3 THE NEW PROBABILISTIC VERIFICATION SCHEME

3.1 Bounding the maximum state omission probability

In probabilistic verification schemes, the verifier only produces false positives if it misses all error states of the protocol. Therefore, if we have a bound on the **maximum state omission probability**, where the maximum is taken over all reachable states, it is also a bound on the probability of the verifier producing false positives. Clearly, the omission probability calculated in the case of hash compaction also provides a bound on the probability of false positives but not a tight one.

In the following analysis, we will assume that breadth-first search is employed for the state space search. In breadth-first search, the states are explored in levels, where states in the ith level are reachable from the startstate in no less than i transitions. Assume that the length of the longest of the shortest paths from the startstate to any reachable state is d, i.e. the diameter of the reachability graph is d. Clearly, the breadth-first search will then explore the levels $0, \ldots, d$.

We now look at an arbitrary state s_d in level d. Let one of the shortest paths from a startstate s_0 to this state be $s_0 s_1 \ldots s_d$, where s_i is a state in the ith level. Intuitively, the omission probability for state s_d is bounded by the omission probability for this path.

More formally, let N_i be the event that state s_i will not be omitted during the search. There are two possible causes for an omission of state s_i. First, the state may be reached but omitted during its insertion into the state table. We call the event that state s_i is not omitted during its **insertion** N_i'. Second, an ancestor of state s_i which was necessary for the reachability of s_i may have been omitted. It follows that

$$N_i \Leftarrow N_i' \wedge N_{i-1}, \text{ for } 0 < i \leq d, \quad \text{and} \quad N_0 \Leftrightarrow N_0' \ .$$

Note that there can be multiple startstates and thus startstates can be omitted as well. It follows for the probabilities of these events that

$$\Pr(N_i) \geq \Pr(N_i' \wedge N_{i-1}) = \Pr(N_i' | N_{i-1}) \Pr(N_{i-1}) \quad \text{and} \quad \Pr(N_0) = \Pr(N_0') \ .$$

Using this result recursively yields

$$\Pr(N_d) \geq \Pr(N_0') \prod_{i=1}^{d} \Pr(N_i' | N_{i-1}) \ . \tag{1}$$

This inequality gives a lower bound on the probability that state s_d is not omitted and thus also an upper bound on the probability that state s_d is omitted. Since we took a state out of the last breadth-first level and the preceding levels have smaller upper bounds, this inequality also also gives us the desired upper bound on the maximum state omission probability. The probabilities on the right-hand side of inequality (1) are dependent on the hash table insertion algorithm and will be approximated in the following discussion.

3.2 Ordered hashing

In the new probabilistic verification scheme, we employ some ideas from hash compaction. Compressed state descriptors instead of the full ones are stored in a hash table and we use open addressing for the collision resolution in this table. However, there is one problem in the previous hash compaction scheme which would become even worse in the new scheme if we used the same algorithm to insert states into the hash table. When the hash table gets almost full, the average length of the probe sequence for finding an empty slot during the state insertion increases sharply. For example, to find the last empty slot in the table, an average of $\frac{m+1}{2}$ probes are needed. The probability of omission during these insertions also increases sharply, since it is roughly proportional to the length of the probe sequences.

For typical protocols, this problem would be exacerbated in the new scheme by the characteristic shape of the distribution of the reachable states over the different levels in the breadth-first search. Usually, this distribution is bell-shaped, i.e. it has few states in the first levels and last levels and the majority in the middle levels. Thus, assuming the table finally fills up, a significant number of states from the 'tail' levels would experience an almost full hash table and thus the sharply increased probability of omission.

The algorithm

Fortunately, there is a hashing scheme whose objective is to reduce the average number of probes needed in unsuccessful searches – ordered hashing introduced by Amble and Knuth (1974). The insertion algorithm in ordered hashing is given in Figure 2. Here, $T[i]$ denotes the ith slot in the hash table (i=0, ..., m−1) and also the value in this slot and h' and h'' are two hash functions which are used to calculate the probe sequence. The hash function h'' has to yield values that are relatively prime to m for the algorithm to work properly. Typically, m is chosen to be prime and h'' yields values between 1 and $m-1$. Since h'' is applied to $c(s)$, there is some dependency between the compressed value and the probe sequence of a state, which has to be taken into account in the analysis of the new scheme.

The algorithm is composed of a search phase and an insertion phase. Note that only the search phase might lead to omissions, since it alone may return 'state already present'. The search loop is left as soon as a slot is found which is either empty **or** contains a value that is **smaller** than the compressed value to be entered. This new way of leaving the search loop strongly reduces the average number of probes for (unsuccessful) searches, especially when the table is relatively full. For example, for a huge m the algorithm only does an average of $\log m$ probes in the search loop if there is one empty slot in the table. The insertion phase maintains the correct arrangement in the table to make the algorithm work properly. Amble and Knuth showed that the average number of probes to insert a state into an ordered hash table is the same as in the conventional hashing case. Only the interchanging of compressed values required by the new algorithm results in a runtime penalty which was, however, negligible in practice.

```
Insert(state s)
begin
  i := h'(s);
  v := c(s);
  while not (T[i] is empty or T[i] < v) do      // search
    if T[i] = v then return 'state already present';
    i := [i + h''(v)] mod m;
  end
  while T[i] is non-empty do      // insertion
    if T[i] < v then interchange the values of T[i] ↔ v;
    i := [i + h''(v)] mod m;
  end
  T[i] := v;
  return 'state inserted';
end
```

Figure 2 Insertion into an ordered hash table

The analysis

Let p_k denote the probability that there is no omission during the insertion of a new state, assuming there are k states already in the table. An approximation formula for p_k assuming $m \gg 1$ can be calculated*, yielding

$$p_k = 1 - \frac{2}{l}(H_{m+1} - H_{m-k}) + \frac{2m + k(m-k)}{ml(m-k+1)} \quad . \tag{2}$$

Here, $H_n = \sum_{i=1}^{n} \frac{1}{i} = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + O(\frac{1}{n^4})$ denotes a harmonic number.

We can use this formula for p_k to give a bound on the probabilities on the right-hand side of (1). Assume that there is a total of k_i states stored in the hash table when the algorithm finishes the ith level. Then, there were at most $k_i - 1$ states in the table when we tried to insert state s_i. Thus, we have the following bounds:

$$\Pr(N_i' | N_{i-1}) \geq p_{k_i - 1}, \text{ for } 0 < i \leq d, \quad \text{and} \quad \Pr(N_0') \geq p_{k_0 - 1} \quad .$$

By plugging these inequalities into (1), we obtain

$$\Pr(N_d) \geq \prod_{i=0}^{d} p_{k_i - 1} \quad . \tag{3}$$

For each level, a bound on the maximum state omission probability for all states explored so far can be calculated 'on-line' (i.e. during the state space search) by the verification algorithm. Finally, the bound on the maximum state omission probability is the value reported in the last level observed.

Note that the last level observed might not be the actual last level of the protocol. Then, the verifier might report too optimistic a value for the maximum state omission

*2-page derivation omitted.

probability. However, this event is relatively rare. Let level t be the last observed level. Then, all states of level $t{+}1$ must have been omitted. Even if level $t{+}1$ has only one state, the omission probability for this state is bounded by $1 - \prod_{i=0}^{t+1} p_{k_i-1}$, where $k_{t+1} := k_t$. This expression is approximately $1 - \prod_{i=0}^{t} p_{k_i-1}$, which was reported by the verifier as the bound on the maximum state omission probability, and which is typically small in practice.

3.3 Choosing the number of bits per state

Let n denote the number of reachable states. For calculating an accurate value of the bound on the maximum state omission probability, the knowledge of the diameter d and the distribution k_i, $0 \le i \le d$, is required. For a coarse approximation, we assume that the table fills up completely ($m{=}n$) and that half the states in the path $s_0 s_1 \ldots s_d$ experience an empty table during their insertion, while the other half experiences a table with only one empty slot. This models (crudely) the typically bell-shaped state distribution over the breadth-first levels $0, \ldots, d$. Assuming further, that the individual probabilities in (3) are close enough to 1 to approximate the product by a sum, we obtain – using (2) and assuming $n \gg 1$ – the following approximate value $\tilde{P_{\text{om}}}$ for the bound on the maximum state omission probability:

$$\tilde{P_{\text{om}}} = \frac{1}{l}\, d\, (\ln n - 1.2) \ . \tag{4}$$

Note that this formula was only derived to give an estimate of the bound on the maximum state omission probability. A real but conservative bound could be calculated by assuming a table with only one empty slot for the whole path $s_0 s_1 \ldots s_d$, which would basically increase $\tilde{P_{\text{om}}}$ by a factor of two.

Comparison of (4) to the formula in Stern and Dill (1995b) for the maximum omission probability in the previous hash compaction scheme, $P_{\text{om}} \approx \frac{1}{l}\, n\, (\ln n - 1)$, shows that we essentially replaced the factor n by the factor d. This corresponds well with the fact that we are now reasoning about paths of maximum length d, and that before we were reasoning about all n reachable states.

The new scheme roughly saves a factor of n/d in the bound on the probability of false positives in comparison to the previous hash compaction scheme, or roughly $\log_2(n/d)$ bits in the compressed state descriptor. Note that in the new scheme, adding just one bit to the compressed values allows squaring the size of the state space or doubling the diameter assuming the maximum state omission probability is to stay the same.

Table 1 shows the diameter, the number of reachable states and the number of bits in the (uncompressed) state descriptors for three industrial protocols when a protocol parameter (the number of processors) is varied. All three protocols – cache3, SCI (IEEE Std 1596–1992; Stern and Dill 1995a) and adash (Lenoski et al. 1992) – are cache coherence protocols. Observe, that the ratio of n/d grows exponentially when the number of processors is increased and that the diameter d is typically quite small.

Table 2 shows the approximate bound $\tilde{P_{\text{om}}}$ on the maximum state omission probability assuming $n{=}10^9$ reachable states while varying the diameter d and the number of bits b for the compressed values. In practice, the given formula for $\tilde{P_{\text{om}}}$ typically yields upper bounds that are more conservative than the bounds reported by the verifier using (3).

Table 1 Diameter d, number of reachable states n and number of bits $\hat{b}$ in the original state descriptor for sample protocols

protocol		procs 2	3	4	5	6	7
cache3	d	16	23	27	31	35	39
	n	299	2796	8534	18 962	37 228	66 296
	$\hat{b}$	87	126	155	179	203	258
SCI	d	22	62	117			
	n	96	18 193	9707 150			
	$\hat{b}$	277	437	582			
adash	d	7	37	n/a			
	n	39	10 466	n/a			
	$\hat{b}$	360	1126	3012			

Table 2 Approximate bounds $\tilde{P_{\text{om}}}$ on the maximum state omission probability for n=10^9 reachable state

b	diameter d 100	200	500
16	3.0%	6.0%	15%
20	0.19%	0.37%	0.93%
24	0.012%	0.023%	0.058%

When using the new verification scheme with b=20 bits per state for protocols with up to 10^9 reachable states and a maximum diameter d of 500, the maximum value for $\tilde{P_{\text{om}}}$ is only 0.93%. Comparing this 20 bits per state with the sizes of the original state descriptors in Table 1 gives, for example, a memory savings factor of 56 for the 3-processor adash protocol.

4 IMPLEMENTATION

The main differences in the implementation between the previous hash compaction scheme and the new one are the on-line calculation of the bound on the maximum state omission probability and the use of ordered hashing. The calculation of the hash and compressed values out of the state descriptor was performed using the H_3 universal$_2$ class of hash functions (Carter and Wegman 1979). More details about this calculation can be found in Stern and Dill (1995b).

In the case of protocol errors, the verifier should be able to provide the protocol designer with an error trace, i.e. a sequence of states leading from a startstate to the error state. When the verifier finds an error using depth-first search, the state queue contains such a

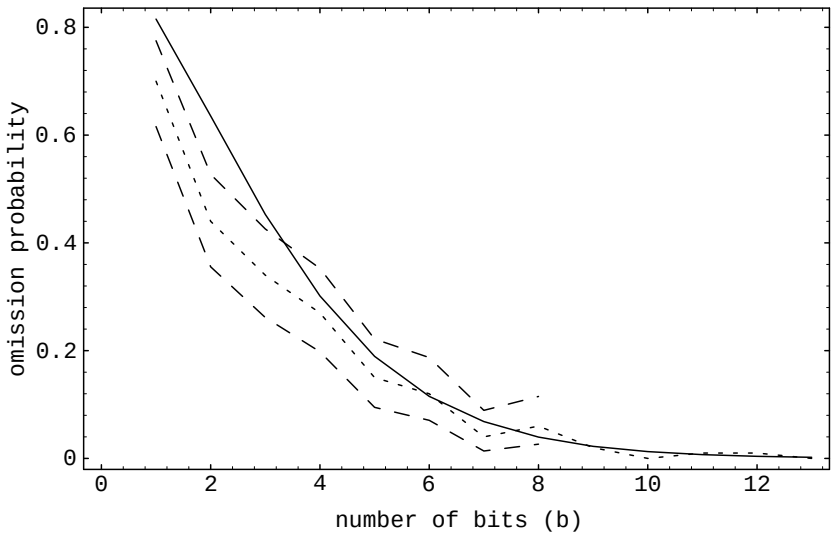

Figure 3 Last state omission probabilities observed in the experiments (dotted, with dashed lines indicating 90% confidence intervals) and bound calculated by the verifier (solid) for the **branching** protocol with n=262 143, d=17 and m=262 147

trace. However, this trace is usually many orders of magnitude longer than the shortest possible trace to expose the error. Breadth-first search, by contrast, can find one of the shortest traces to an error, but requires additional information to be stored.

In the previous Murφ implementation, a pointer to a predecessor state was stored together with each state. However, these pointers would now require more memory than the compressed values. A solution to this problem is writing a record – consisting of the index of the predecessor's record and the compressed value – into a file for each new state. This information is sufficient to generate an error trace. Clearly, the file is sequentially accessed; in practice, no slow-down has been observed due to maintaining this file.

5 RESULTS ON SAMPLE PROTOCOLS

For all the experiments described in the following, we varied the number of bits b in the compressed values. For each value of b, we conducted 100 runs of the verifier and counted the number of runs in which a particular state was omitted. For this state, we chose the last state that is explored in a breadth-first search without omissions. Clearly, this state is in the last breadth-first level and should have an omission probability that approximately equals the bound on the maximum state omission probability reported by the verifier, assuming there are not too many paths to this state. If there are many paths to the last state, one would expect the observed omission probability to be smaller than the reported bound.

5.1 An artificial protocol

Figure 3 shows the outcomes of our experiments on an artificial 'branching' protocol, in which each state – except the ones in the last level – has two children which are different

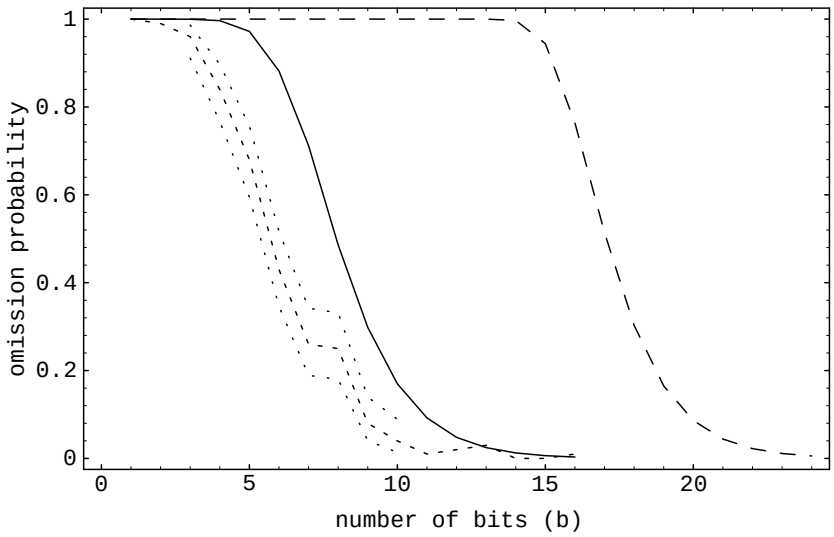

Figure 4 State omission probabilities observed in the experiments (dotted, with 90% confidence intervals) and bound calculated by the verifier (solid) and omission probabilities in the previous scheme (dashed) for the **SCI** protocol with n=18 193, d=62 and m=18 229

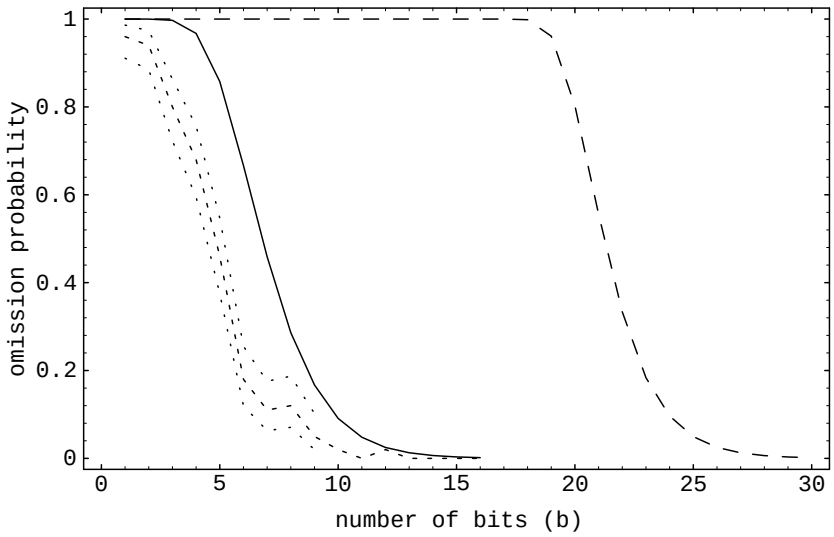

Figure 5 State omission probabilities observed in the experiments (dotted, with 90% confidence intervals) and bound calculated by the verifier (solid) and omission probabilities in the previous scheme (dashed) for the **cache3** protocol with n=200 913, d=27 and m=200 927

from all other states. Thus, there is only a single path to each state and as a result the observed last state omission probabilities are close to the calculated bounds. The difference between the two curves for bigger values of the probabilities can be explained by observing that the (omitted) analysis of p_k is conservative in this case.

5.2 Industrial protocols

Figures 4 and 5 show the outcomes for instances of the protocols SCI and cache3, respectively. Note, that the ratio of n/d is bigger for the cache3 protocol and so are the savings in the number of bits in comparison to the previous scheme. Here, the cache3 protocol was verified without using symmetry reductions (Ip and Dill 1993), hence the difference in the number of reachable states in comparison to Table 1. Furthermore, note that the observed last state omission probabilities are not much smaller compared to the reported bounds, although there might now be many paths in the protocols leading to this state. This might be due to the fact that if the different paths leading to a state contain common states, the path omission probabilities will strongly depend on each other and almost no reduction in the omission probability of the state will occur.

6 CONCLUSION AND FUTURE WORK

The new probabilistic verification scheme presented in this paper reduces the number of bits that have to be stored per state for reliable verification – in which the probability of false positives is bounded. For protocols with up to 10^9 reachable states and a maximum diameter of 500, 20 bits per state are sufficient to guarantee a bound on the probability of false positives of 0.93%.

Bitstate hashing, in contrast, does not guarantee any bound on the probability of false positives, but is also usable with less than 20 bits per state. At the beginning of a verification project, the verifier typically finds errors, and the lack of a bound on the probability of false positives in bitstate hashing does not matter. However, one typically starts verifying down-scaled protocol models (Dill et al. 1992), for which probabilistic verification schemes are not really needed. Later, when the models become bigger and the verifier usually finds no more errors (produces positives), the new scheme is advantageous, since it provides a bound on the probability of false positives.

In comparison to the previous hash compaction scheme, the memory savings are achieved primarily by calculating a tighter bound on the probability of false positives. Recall that this bound can be controlled by selecting the number of bits b stored per state. Furthermore, re-running the verifier with independent hash and compression functions allows multiplication of the reported bounds.

The new scheme is compatible with several methods that aim at reducing the size of the reachability graph while ensuring that errors will still be detected. Examples would be exploiting symmetries (Ip and Dill 1993) and reversible rules (Ip and Dill 1996). When combining different techniques, one usually observes that runtime becomes the new major limiting factor in verification, which increases the priority of research into accelerating explicit-state verification methods.

When using a depth-first traversal of the state-space, one recommendation is to use the t-limited hashing scheme presented in Stern and Dill (1996) – which limits the length of the probe sequences to t probes – instead of ordered hashing. Then, a formula for a bound P_{om} on the maximum state omission probability can be derived, yielding $P_{\text{om}} = \frac{1}{l}\, d' t$, where d' is the maximum depth that occurred during the depth-first search. Unfortunately, this depth is often much larger than the diameter of the state space and thus the savings of

the new scheme are usually smaller than in the breadth-first case. However, techniques to reduce d' or a bound on the diameter can be employed in the depth-first case.

ACKNOWLEDGEMENTS

We are grateful to Pierre Wolper and Denis Leroy for sharing the unpublished revision (Wolper and Leroy 1995) of their paper with us. We would also like to thank Ravi Soundararajan for his comments on a draft of this paper. Ulrich Stern was supported during this research by a scholarship from the German Academic Exchange Service (DAAD-Doktorandenstipendium HSP-II).

REFERENCES

O. Amble and D. E. Knuth (1974) Ordered hash tables. *Computer Journal*, 17(2):135–42.

J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill (1990) Sequential circuit verification using symbolic model checking. In *27th ACM/IEEE Design Automation Conference*, pages 46–51.

J. L. Carter and M. N. Wegman (1979) Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–54.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest (1990) *Introduction to Algorithms*. The MIT Press.

D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang (1992) Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–5.

G. J. Holzmann (1987) On limits and possibilities of automated protocol analysis. In *Protocol Specification, Testing, and Verification. 7th International Conference*, pages 339–44.

G. J. Holzmann (1991) *Design and Validation of Computer Protocols*. Prentice-Hall.

G. J. Holzmann (1995) An analysis of bitstate hashing. In *Protocol Specification, Testing and Verification. 15th International Conference*, pages 301–14.

A. J. Hu, G. York, and D. L. Dill (1994) New techniques for efficient verification with implicitly conjoined BDDs. In *31st Design Automation Conference*, pages 276–82.

IEEE Std 1596-1992, IEEE Standard for Scalable Coherent Interface (SCI).

C. N. Ip and D. L. Dill (1993) Better verification through symmetry. In *11th International Conference on Computer Hardware Description Languages and their Applications*, pages 97–111.

C. N. Ip and D. L. Dill (1996) State reduction using reversible rules. In *33rd Design Automation Conference*.

D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam (1992) The Stanford Dash multiprocessor. *Computer*, 25(3):63–79.

U. Stern and D. L. Dill (1995a) Automatic verification of the SCI cache coherence protocol. In *IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 21–34.

U. Stern and D. L. Dill (1995b) Improved probabilistic verification by hash compaction. In *IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 206–24.

U. Stern and D. L. Dill (1996) Combining state space caching and hash compaction. In *Methoden des Entwurfs und der Verifikation digitaler Systeme, 4. GI/ITG/GME Workshop*, pages 81–90.

P. Wolper and D. Leroy (1995) Reliable hashing without collision detection. Unpublished revised version of Wolper and Leroy (1993).

P. Wolper and D. Leroy (1993) Reliable hashing without collision detection. In *Computer Aided Verification. 5th International Conference*, pages 59–70.