International Journal of Computer and Communication Technology

Volume 7 | Issue 4 Article 1

October 2016

COVERAGE INFERENCE IN MODEL CHECKING USING SEQUENTIAL MULTIPLE HASHING

T. SUGANYA
Computer Science, Anna University, Sona College of Technology, Salem, India., suganya.t.raju@gmail.com

T. SATHIYA
Computer Science, Anna University, Sona College of Technology, Salem, India., hemusathya@gmail.com

COVERAGE INFERENCE IN MODEL CHECKING USING SEQUENTIAL MULTIPLE HASHING

T.SUGANYA & T.SATHIYA

Computer Science, Anna University, Sona College of Technology, Salem, India.
Email: suganya.t.raju@gmail.com, hemusathya@gmail.com

Abstract:-Model checking automatically tests whether a model meets a given specification or not. It is a technique for verifying correctness properties of finite-state systems. One of the major problems in model checking is the state-explosion. To overcome this, a probabilistic approach called Bit-state Hashing is used to reduce the memory requirements. Bit-state hashing uses a data structure called bloom filter to store the corresponding reached states in a hash table. By enlarging a bloom filter, it improves total coverage estimation using a growth curve that approximates increased in reached states. To increase the effectiveness of the existing system, coverage estimation in model checking with sequential multiple bit-state hashing has been proposed. The sequentially repeated bit state hashing technique can outperform all other hashing methods, even for very large problem size.

Index Terms- *Model Checking, Hashing Techniques, Coverage Estimation*

I. INTRODUCTION

Model checking [1][3][4] verifies any kind of application or system design that the given model meets a client specification or not. It's a technique for automatically verifies the correctness properties of a system. Model checking is costly in terms of three factors : time, memory, modelling effort. One of the problems in model checking is the state explosion problem. In order to avoid the state explosion problem, a probabilistic approach bitstate hashing technique is used.

Memory required for state space is depends on the number of reached states of that particular model. Therefore if the model level is very high, then there will be a need of high memory requirement [6]. For this purpose a bitstate approach [2] is used for state hashing, in this each state is translated into a numerical bits and it is passed to hash table by enlarging a bloom filter [9].
The bloom filter consist of two fundamental actions: *add* (adding an element to a bloom filter) and *query* (checking whether an element is a member or not), all bits at which k hash values point are verified. Then the resulting function is taken as the index to an array of bit-field or bits, if the state is already reached it is represented as 1 or else stores 1 by itself if not already reached. Explicit state model checking and state model checking are the two types of algorithms involved in model checking process.

In Bitstate hashing [2] there is a chance for occurrence of collision in between states. To overcome this, a memory needed for keeping the table of visited states should be reduced. Initially all the bit of table is set to 0. To determine whether the states is in the table, hash function can be applied to the state and it checks whether the corresponding bit is at the computed address is 1.The main drawback of this method is that the hash function may compute the same address for unique states and can wrongly results that a state has already visited. Alternatively, one may use two hash functions and stores two bits in the table for each state and concludes that a state is present in the table, where if the bits are computed at both hash functions and are set to 1.Then the coverage is estimated using different hash functions. If the actual coverage is low, then it results in difficulty in coverage estimation for bitstate hashing. Therefore it is difficult to achieve high accuracy from the limited amount of information.

To overcome the collision detection problem[5] in Bitstate hashing technique, a sequential multiple hashing techniques[8] is proposed, in this performing reachability analysis with single bit or double bit hashing repeatedly, each time with a different and independent hash function.

II. RELATED WORK

A. Bitstate Hashing Analysis

The Bitstate hashing, or super-trace [2] is a method to increase the quality of verification by reachability analyses for applications that defeat analysis by traditional means because of their size. Since then, the technique has been included in many research verification tools, and was adopted in tools that are marketed commercially. It is therefore important that we understand well how and why the method works, what its limitations are, and how it compares with alternative methods over a broad range of problem sizes. The original motivation of bitstate hashing technique was based on empirical evidence of its effectiveness. In this paper they provide an analytical argument and then compare the technique with two alternatives that have been proposed in the recent literature. They also describe a sequential bitstate hashing technique that can be of value when confronted with very large problem sizes. The goal of the bitstate hashing technique is to minimize the loss and maximize the coverage. By considering two cases separately: $N<M$ and $N\geq M$.

In case of single bit hashing: When N>M, the first state for that entering a value into the bit array has zero probability of collision of
(N-1)/M.

$$P_1 \leq \frac{1}{N-M}\sum_{i=0}^{N-1} i = \frac{N-1}{2M} \leq 2^{n-m-1}$$

The expected coverage of the search is(1-P_1).100%,so P_1 will be as small as possible.
When N≥M, i.e., the number of states is larger than the number of available bits,(N-M) will be the small number of hash collisions and hence the smallest possible value for the average probability of collision becomes:

$$P_1 \geq 1 - \frac{M}{N} = 1 - 2^{m-n}$$

The simplest approximation will be more accurate as N>>M or n>>m.
In case of Multi-bit Hashing,
Performance of single-bit hashing is enhanced by using multiple hash functions. It uses two hash functions, and correspondingly stores two bits in the array for each and every state. When h.N≥M, the average probability of collisions can therefore be:

$$P_h \leq 1 - \frac{M}{h.N} = 1 - \frac{1}{h}2^{m-n}$$

The hash functions *h* two kinds of effects, they are:

1. It increases the coverage for higher values of M(maximum amount of available memory), and decreases it for lower values.
2. It reduces range of values of M, for which optimal coverage is reached.

In Sequential Multi-hash technique, if the reachability graph is well connected, then there will be of unique paths that leads to the same state in the graph, and the truncation of a minimum number of paths due to hash-collisions cannot prevent states from the paths being reached.

B. Bloom Filters in Probabilistic Verification

Hash Compaction and bitstate method [9] are the leading techniques to store states in a bloom filter. The main goal is that introducing large dependences among the hash functions of a Bloom filter. A probabilistic verification approaches utilize following two data structures: Bloom filter and compacted hash table. By having three kind of probabilistic techniques, each of which trust is the best selection for some level of familiarity of the state space size.

1. when the state space size is completely unknown,
2. when we know the size of the state space rather than accurately,
3. when we have a rough estimate of the state space size.

In general, bloom filter is used to characterize subsets of a few universe U. A Bloom filter is implemented as an array of m bits, uses k index functions mapping elements in U to(0..m),and supports two basic operations: add and query. To add the element in a Bloom filter, the index functions are used to generate k indices into the array and the consequent bits are set to 1.A query is positive if and only if all k referenced bits are 1.A negative query clearly indicates that the element is not in the bloom filter, but a positive query could be due to a false positive. As a result, bloom filter turns into exacting state space size estimates runs at speeds approaching a super-trace bloom filter.

C. Reliable Hashing Without Collision Detection

Hashing without collision detection [5] is a way to reduce the memory required to store the explored state space. This is done by under some conditions, can reliably analyze a system with many fewer states. Consider the following problem: Given a program P represented by an initial state s_0 and a function succs(s) which yields the set of immediate successors of any state s. Our purpose is to focus on the search process. The algorithm used for the search is described below: where the variable Stack denotes a stack structure and variable T denotes a lookup table.

1. Initialize: Stack: = [s_0]; T :={s_0};

2. Loop: while Stack≠ ∅ do

```
begin
  s:=pop(Stack);
  for all s' ∈ succs(s)do
      begin
      if  s' ∉T then
        begin
        insert s' ∉in T;
        push s' onto Stack;
        end
      end
end
```

The memory requirements of this algorithm are stack and a table. The stack is consecutively accessed and has its length bounded by the depth of the state-space graph. On the other hand, the table has to allow direct access to its elements and will eventually have the whole state-space.

D. Lurch

Here comparing the performance of Lurch to the popular tools *h* such as SMV and SPIN [3]. Model checking is a powerful one, however it could be costly:

1 State-space Explosion problem: the single global finite-state machine constructed to represent all

possible interactions of the local machines requires too much memory.

2 In order to minimize the global state space, the size and features of input models is restricted.

SMV (Symbolic Model Verifier), uses BDD's (Binary Decision Diagrams), to symbolically represent the global state space. SPIN is a model checking tool, uses partial order reduction and mainly targets asynchronous models.In comparison to SMV and SPIN, Lurch performed Well. Only problem using Lurch is a false positive result.

E. Swarm Verification Techniques

Swarm [10] is a one kind of tool that allows the user to search randomization and diversification options. The tool is built as a script generator for verification process and this will generate a shell script that execute as many different types of verification runs as likely without exceeding user defined constraints on time and memory use. Time and Memory constraints are tightly connected; the tool only desires to take the least amount of these two constraints into consideration. The search method described in this can be extended in different ways. This tool also applicable for very large problem sizes contained by user defined time or memory bounds.

III. PROPOSED WORK

To overcome the problem of bitstate hashing approach, sequential multiple hashing technique is proposed. It performs reachability analysis using single-bit or double-bit hashing repeatedly, each time with a unique and independent hash functions.

The hash collision occurs between unique states in each of an initial series, there should exist an independent hash-function that divides these states.

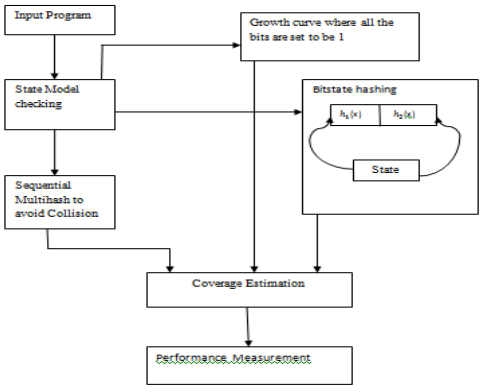

Fig 1: System Architecture

Using multi-hash function, collision crisis will be avoided. At the first run chance of collision is ***p*,** beneath ideal circumstances it decreases to . A fine selection for h for sequential Bitstate hashing, is again a trade-off between expected precision and runtime costs.

IV CONCLUSION

Thus to conclude that when comparing to bitstate hashing approach, sequential multiple hashing carry out well. The collision can be avoided using sequential hashing technique. It performs reachability analysis using single-bit or double-bit hashing repeatedly, each time with a unique and independent hash functions. The sequentially repeated bit state hashing technique can do better than all other hashing methods, even for very large problem size. In future additional parameters might be necessary for coverage inference.

REFERENCES

[1] S. Ikeda, M. Jibiki, Y. Kuno, "Coverage Estimation in Model Checking With Bitstate Hashing," *IEEE Computer Society*, 2012.

[2] G. J. Holzmann, "An Analysis of Bitstate Hashing," *Formal Methods in System Design*, vol. 13, no. 3, pp. 289–307, 1998.

[3] G. J. Holzmann, *The Spin Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.

[4] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model Checking Programs," *Automated Software Engineering*, vol. 10, no. 2, pp. 203–232, 2003.

[5] P. Wolper and D. Leroy, "Reliable Hashing without Collision Detection," in *Proceedings of 5th International Conference on Computer Aided Verification*, 1993, pp. 59–70.

[6] U. Stern and D. L. Dill, "A New Scheme for Memory-efficient Probabilistic Verification," in *IFIP TC6/WG6.1 Joint International Conference on Formal Description Technique for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification*, 1996, pp. 333–348.

[7] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp.422–426, 1970.

[8] J. Eckerle and T. Lais, "New Methods for Sequential Hashing with Supertrace," 1998.

[9] P. C. Dillinger and P. Manolios, "Bloom Filters in ProbabilisticVerification," *Lecture Notes in Computer Science*, vol. 3312, pp. 367–381, 2004.

[10] G. Holzmann, R. Joshi, and A. Groce, "Swarm Verification Techniques,"*IEEE Transactions on Software Engineering*, vol. 37, no. 6,pp. 845–857, 2011.

[11] D. Owen and T. Menzies, "Lurch: a Lightweight Alternative to Model Checking," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, 2003, pp. 158–165.

[12] R. Pela´nek, T. Hanzˇl, I. Cˇ erna´, and L. Brim, "Enhancing Random Walk State Space Exploration," in *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems*,2005, pp. 98–105.

[13] R. Grosu and S. A. Smolka, "Monte Carlo Model Checking," *Toolsand Algorithms for Construction and Analysis of Systems (TACAS 2005)*, vol. 3440, pp. 271–286, 2005.

[14] E. Tronci, G. D. Penna, B. Intrigila, and M. V. Zilli, "A Probabilistic Approach to Automatic Verification of Concurrent Systems," *Asia-Pacific Software Engineering Conference*, pp. 317–324, 2001.

◈◈◈