A Simple State-Space Reduction Method

Thomas Mailund April 10, 2004

We present a very simple approach to reducing the memory usage during a complete state space exploration. The method guarantees to examine each reachable state at least once, but potentially several times, thus choosing memory performance over time performance in a time/memory trade-off.

INTRODUCTION

We are often interested in finding out if a given predicate on states of a system can be satisfied in any or all of the systems reachable states. To determine this, one approach is to calculate the complete set of reachable states and evaluate the predicate on each of these states. To know which states we have already examined and which we have yet to examine—and thus knowing when we have examined all states—we store the examined states in a table.

The major problem with this approach is the so-called state explosion problem: that the number of reachable states of a system increases dramatically with the complexity of the system, making it impractical to store all reachable states in memory. A number of approaches have been developed to overcome this problem, ranging from efficient representations of the individual states and sets of states [6, 8] to techniques for only calculating some of the state space, while still preserving the properties under interest [4, 10]; deleting states after examining them [1, 5]; storing only representatives for classes of states [2, 3]; or relying on probability theory to examine all states with high probability, without explicitly representing the set of states [7, 9].

Here we propose a new reduction method that uses randomisation to decide when to store a state in our table and when to discard it after examining it. The method is guaranteed to examine all reachable states at least once.

A SIMPLE REDUCTION METHOD

The method is based on the simple idea of only storing a new state in our set of known states with a certain probability; in essence we flip a coin to determine if the state should be stored or forgotten. Since we do not store all states right away, we hope to save memory—if we store states with probability p we expect to have stored $p \cdot N$ states after having examined N states—but of course at the cost of potentially re-examining the same

states several times, if several paths leads to the same state or if the system has cyclic behaviour. If the same state is examined several times, however, it will eventually be stored in our set of states—if states are stored with probability p we expect to examine a state $1/p$ times before storing it—so the method will eventually terminate. Since we do not terminate before we know that there are no unexamined states, we guarantee that each state is examined at least once.

One problem with the proposed method is that it sacrifices runtime for memory usage; after having examined N states it will only have stored $p \cdot N$ of them, but because of this it might not recognise a previously examined state if it reaches it again, and it will therefore re-examine it and continue to examine its successors who have also already been examined. For small values of p we will only store a small fraction of the examined states, so we potentially have to reexamine the same states a large number of times before finally storing them, leading to a significantly increase in runtime. This is especially a problem since the parameter p of the method is selected *a priori*, when an appropriate value might not be obvious; if we select p too high, we risk storing many more states than needed, but if we select p too low, we risk having to examine large parts of the state space several times.

To alleviate this problem somehow, we can let p be a function of time, or the number of states examined so far. If p starts out small, we will not store a lot of states unnecessarily if we quickly terminate, but as we explore more and more states, using more and more time, we will start storing more and more of the states seen, until we eventually will store all new states until we finally terminate. This does not completely eliminate the problem of choosing an appropriate p, now as a function rather than a fixed probability, but at least the probability will adapt to the problem somewhat.

A variation of the idea above is to let p *decrease*, as a function of current memory usage. If the memory begins to fill up, it might be appropriate to save a smaller fraction of the examined states. This, again, increases the running time, but the alternative is exhausting available memory, leading to paging which also significantly increases the running time.

EXPERIMENTS

To evaluate the method experimentally, we need to select a number of systems with various kinds of state space graphs (narrow/wide, strongly connected or DAGs, complete/sparse), run the method on the systems with various choices of parameter p, and obtain mean and standard deviation.

REFERENCES

1. S. Christensen, L.M. Kristensen, and T. Mailund, *A Sweep-Line Method for State Space Exploration*, Proc. of TACAS'01, LNCS, vol. 2031, Springer-Verlag, 2001, pp. 450–464.
2. E. Clake, E.A. Emerson, S. Jha, and A.P. Sistla, *Symmetry Reductions in Model Checking*, Proc. of CAV'98, LNCS, vol. 1427, Springer-Verlag, 1998, pp. 147–159.
3. E.A. Emerson and A.P. Sistla, *Symmetry and Model Checking*, Formal Methods in System Design **9** (1996).
4. P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems—An Approach to the State-Explosion Problem*, LNCS, vol. 1032, Springer-Verlag, 1996.
5. P. Godefroid, G.J. Holzmann, and D. Pirottin, *State-Space Caching Revisited*, Formal Methods in System Design **7** (1995), no. 3, 227–241.
6. G.J. Holzmann, *State Compression in SPIN: Recursive Indexing and Compression Traning Runs*, Proc. of 3rd SPIN Workshop, 1997.
7. _____, *An Analysis of Bitstate Hashing*, Formal Methods in System Design **13** (1998), 289–307.
8. A. Puri and G.J. Holzmann, *A Minimized Automaton Representation of Reachable States*, Journal on Software Tools for Technology Transfer **2** (1999), 270–278.
9. U. Stern and D.L. Dill, *Improved Probabilistic Verification by Hash Compaction*, Correct Hardware Design and Verification Methods, LNCS, vol. 987, Springer-Verlag, 1995, pp. 206–224.
10. A. Valmari, *A Stubborn Attack on State Explosion*, Proc. of CAV'90, LNCS, vol. 531, Springer-Verlag, 1990, pp. 156–165.