Parallel Recursive State Compression for Free

Alfons Laarman, Jaco van de Pol, and Michael Weber

Formal Methods and Tools, University of Twente, The Netherlands
{a.w.laarman,vdpol,michaelw}@cs.utwente.nl

Abstract. This paper focuses on reducing memory usage in enumerative model checking, while maintaining the multi-core scalability obtained in earlier work. We present a multi-core tree-based compression method, which works by leveraging sharing among sub-vectors of state vectors.

An algorithmic analysis of both worst-case and optimal compression ratios shows the potential to compress even large states to a small constant on average (8 bytes). Our experiments demonstrate that this holds up in practice: the median compression ratio of 279 measured experiments is within 17% of the optimum for tree compression, and five times better than the median compression ratio of SPIN’s COLLAPSE compression.

Our algorithms are implemented in the LTSmin tool, and our experiments show that for model checking, multi-core tree compression pays its own way: it comes virtually without overhead compared to the fastest hash table-based methods.

1 Introduction

Many verification problems are computationally intensive tasks that can benefit from extra speedups. Considering recent hardware trends, these speedups do not come automatically for sequential exploration algorithms, but require exploitation of the parallelism within multi-core CPUs. In a previous paper, we have shown how to realize scalable multi-core reachability [14], a basic task shared by many different approaches to verification.

Reachability searches through all the states of the program under verification to find errors or deadlocks. It is bound by the number of states that fit into the main memory. Since states typically consist of large vectors with one slot for each program variable, only small parts are updated for every step in the program. Hence, storing a state in its entirety results in unnecessary and considerable overhead. State compression solves this problem, as this paper will show, at a negligible performance penalty and with better scalability than uncompressed hash tables.

Related work. In the following, we identify compression techniques suitable for (on-the-fly) enumerative model checking. We distinguish between generic and informed techniques.
Generic compression methods, like Huffman encoding and run length encoding, have been considered for explicit state vectors with meager results [12, 9]. These entropy encoding methods reduce information entropy [7] by assuming common bit patterns. Such patterns have to be defined statically and cannot be “learned” (as in dynamic Huffman encoding), because the encoding may not change during state space exploration. Otherwise, desirable properties, like fast equivalence checks on states and constant-time state space inclusion checks, will be lost.

Other work focuses on efficient storage in hash tables [6, 10]. The assumption is that a uniformly distributed subset of $n$ elements from the universe $U$ is stored in a hash table. If each element in $U$ hashes to a unique location in the table, only one bit is needed to encode the presence of the element. If, however, the hash function is not so perfect or $U$ is larger than the table, then at least a quotient of the key needs to be stored and collisions need to be dealt with. This technique is therefore known as key quotienting. While its benefit is that the compression ratio is constant for any input (not just constant on average), compression is only significant for small universes [10], smaller than we encounter in model checking (this universe consists of all possible combinations of the slot values, not to be confused with the set of reachable states, which is typically much smaller).

The information theoretical lower bound on compression, or the information entropy, can be reduced further if the format of the input is known in advance (certain subsets of $U$ become more likely). This is what constitutes the class of informed compression techniques. It includes works that provide specialized storage schemes for certain specific state structures, like petri-nets [8] or timed automata [17]. But, also COLLAPSE compression introduced by Holzmann for the model checker SPIN [11]. It takes into account the independent parts of the state vector. Independent parts are identified as the global variables and the local variables belonging to different processes in the SPIN-specific language PROMELA.

Blom et al. [1] present a more generic approach, based on a tree. All variables of a state are treated as independent and stored recursively in a binary tree of hash tables. The method was mainly used to decrease network traffic for distributed model checking. Like COLLAPSE, this is a form of informed compression, because it depends on the assumption that subsequent states only differ slightly.

Problem statement. Information theory dictates that the more information we have on the data that is being compressed, the lower the entropy and the higher the achievable compression. Favorable results from informed compression techniques [8, 17, 11] confirm this. However, the techniques for petri-nets and timed automata employ specific properties of those systems (a deterministic transition relation and symbolic zone encoding respectively), and, therefore, are not applicable to enumerative model checking. COLLAPSE requires local parts of the state vector to be syntactically identifiable and may thus not identify all equivalent parts among state vectors. While tree compression showed more impressive