Incremental Hashing for SPIN

Viet Yen Nguyen\textsuperscript{1} and Theo C. Ruys\textsuperscript{2}

\textsuperscript{1} RWTH Aachen University, Germany
\textsuperscript{2} University of Twente, The Netherlands

http://www-i2.cs.rwth-aachen.de/~nguyen/
http://www.cs.utwente.nl/~ruys/

Abstract. This paper discusses a generalised incremental hashing scheme for explicit state model checkers. The hashing scheme has been implemented into the model checker SPIN. The incremental hashing scheme works for SPIN’s exhaustive and both approximate verification modes: bitstate hashing and hash compaction. An implementation is provided for 32-bit and 64-bit architectures.

We performed extensive experiments on the BEEM benchmarks to compare the incremental hash functions against SPIN’s traditional hash functions. In almost all cases, incremental hashing is faster than traditional hashing. The amount of performance gain depends on several factors, though.

We conclude that incremental hashing performs best for the (64-bits) SPIN’s bitstate hashing mode, on models with large state vectors, and using a verifier, that is optimised by the C compiler.

1 Introduction

An explicit state model checker is a model checker where all states are explicitly represented in the state space. Explicit model checking is sometimes called stateful state space exploration, especially when checking reachability or safety properties (e.g. deadlocks, assertion violations).

Central to stateful state space exploration is the process of state matching: for every encountered state, it should be checked whether the state has already been visited or not. As the run-time of exploration is linear in the number of transitions (i.e. the amount of newly encountered states and re-visited ones), it is obvious that state matching should be as fast as possible. Typically, hash tables are used to store states. Upon exploration of each state, the hash table is consulted to check whether that state has already been explored or not.

The access to a hash table is through a hash function. Given a key $k$, a hash function $h$ computes the hash code $h(k)$ for this key. This hash code $h(k)$ corresponds to the address in the hash table, where this key $k$ should be stored. For model checking, this $k$ is typically the (binary) representation of a state, called the state vector. Most traditional hash functions compute $h(k)$ by considering all elements of $k$. For example, if $k$ is a string, a typical hash function $h$ would compute $h(k)$ on the basis of all individual characters of $k$. 

With respect to state space exploration, two observations can be made. Firstly, the size of a state vector is usually substantial. State vectors of several hundreds of bytes are not exceptions. This means that computing a traditional hash code for such states can become quite expensive. Secondly, when exploring the state space in a structured manner (e.g. depth first search), the transitions between two consecutive states is local: only a small part of the state changes with respect to the previous state.

This last observation is the idea behind so called incremental hash functions, which use the hash code of a previous key to compute the hash code for the new key. The application of incremental hashing within a model checker is not new. Mehler and Edelkamp [11] implemented an incremental hashing scheme in the model checker StEAM, a model checker for C++. However, their incremental hashing scheme is only practicable for hashing (large) stacks and queues incrementally. We have improved their hashing scheme by generalising it for hashing vector-based data structures (like state vectors) incrementally by using cyclic polynomials from [2].

This improved scheme was originally developed for MoonWalker [14], a software model checker for CIL bytecode programs, i.e. .NET applications. Unfortunately, after implementing our incremental hash function into MoonWalker, initial tests showed no measurable performance gain. We studied this observation using a profiler and found out that the stake of hashing in MoonWalker is extremely small [12]. Other tasks that have to be performed for each state (e.g. garbage collection, state compression, etc.) take much more time. Any performance gain in hashing would therefore not be visible in the total running time.

The model checker SPIN [6] is arguably one of the fastest explicit state model checkers available. The current version of SPIN uses two traditional hash functions: one composed by Bob Jenkins [20] and one composed by Paul Hsieh [19]. For SPIN verifiers – unlike for bytecode model checkers – hashing accounts for a substantial amount of the running time.

We have implemented our generalised incremental hashing scheme into SPIN 5.1.4. The incremental hashing scheme works for checking safety properties (-DSAFETY) for both 32-bit and 64-bit architectures. Furthermore, it works in exhaustive mode and both approximate modes: bitstate hashing and hash compaction. We performed numerous experiments on the BEEM benchmarks [16] to compare the incremental hash functions against SPIN’s traditional hash functions. From these experiments we learnt that incremental hashing is faster than SPIN’s traditional hash implementations without sacrificing too much accuracy.

The amount of performance gain depends on several factors:

- the verification mode: exhaustive or approximate,
- the architecture on which the verification is run: 32-bit or 64-bit,
- the size of the state vector, and

---

1 MoonWalker was previously known as MMC: the Mono Model Checker. Due to several name clashes, MMC has recently been renamed to MoonWalker.