Bit-state caching

or

Speeding up model checking by modifying hash functions

Juhan Ernits

Koke, 03.02.2006*

* With readable fonts and references.

What do we want to do?

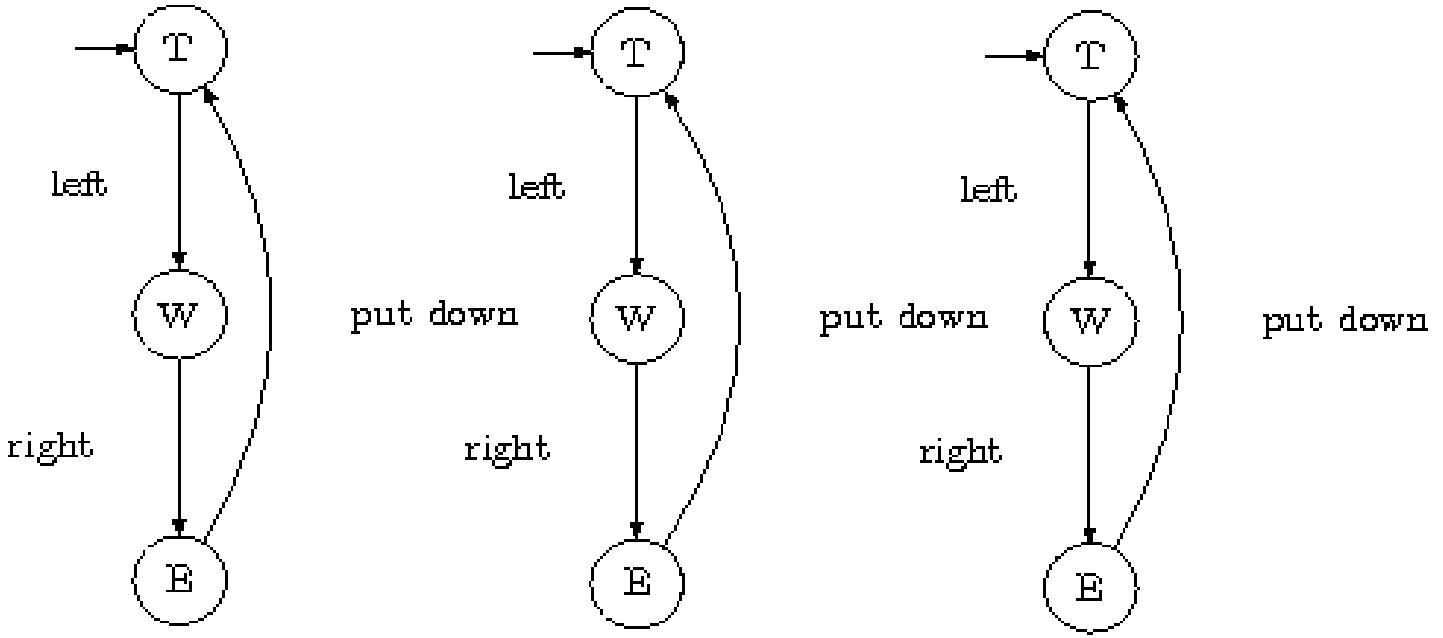
- We want to check for reachability on a structure representing a constraint system.
- (in other words) We want to check if the behaviour of the model is included in the behaviours of the specification

Right! but, really?

- We want to detect if deadlocks are possible in certain software;
- We want to synthesize certain hardware components (for example memory arbiters);
- We want to generate tests from models;
- We want to solve logistics related problems.

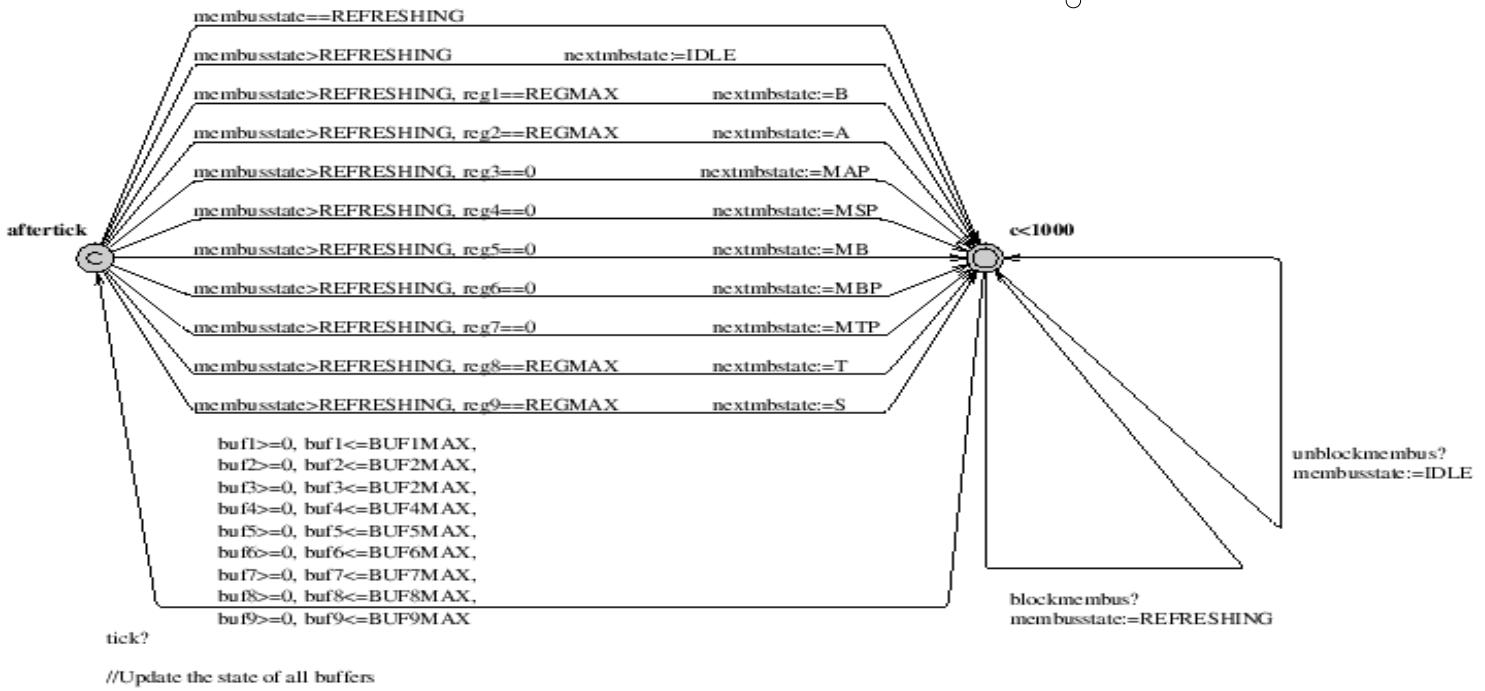
Academic Example

Dining philosophers

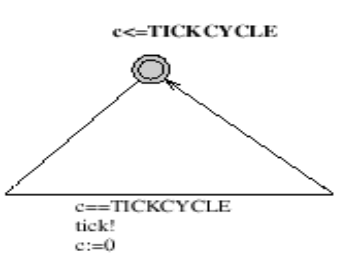


Can this system deadlock?

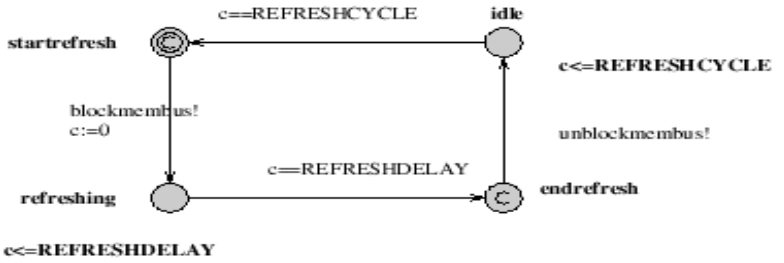
another Example



Buffers



Clock



MemoryRefresh

[Ernits, Memory arbiter synthesis and verification, 2005]

Explicit state model checking

- We consider explicit state model checking.
  - all control states and data states are represented explicitly.
- As opposed to symbolic model checking
  - where the states are represented by some symbolic construct, for example BDD-s.

Ways of reducing memory consumption

- Partial order reduction
- Lossless state compression
  - Collapse compression
  - Minimized automaton representation
- Lossy state compression
  - bit-state hashing
  - hash compaction

Collapse compression

- The state explosion is due to small changes in many places
- Store different parts of the state space in separate descriptors and represent the actual state as an index to relevant state descriptors

Minimized automaton representation

- Build a recognizer automaton for states. All states that have been seen lead to an accepting state.
- The recognizer automaton is interrogated on each step of the model checker.
- The recognizer automaton is modified each time a new state is seen.

What is hash compaction

- A method where each state is represented by a hash (for example 128 bits). This is stored in a regular hash table.
- Used in Spin, Zing, Bogor, ...
- Can achieve very good coverage.

Bit-state hashing

- Let us look at how a hash table works.
- Instead of a state, store one bit.

Hash functions

- mod sucks! (they say)
- Look at Jenkins' hash funcion:

```
// Most hashes can be modelled
// like this:

  initialize(internal state)
  for (each text block)
  {
    combine(internal state, text block);
    mix(internal state);
  }
  return postprocess(internal state);
```

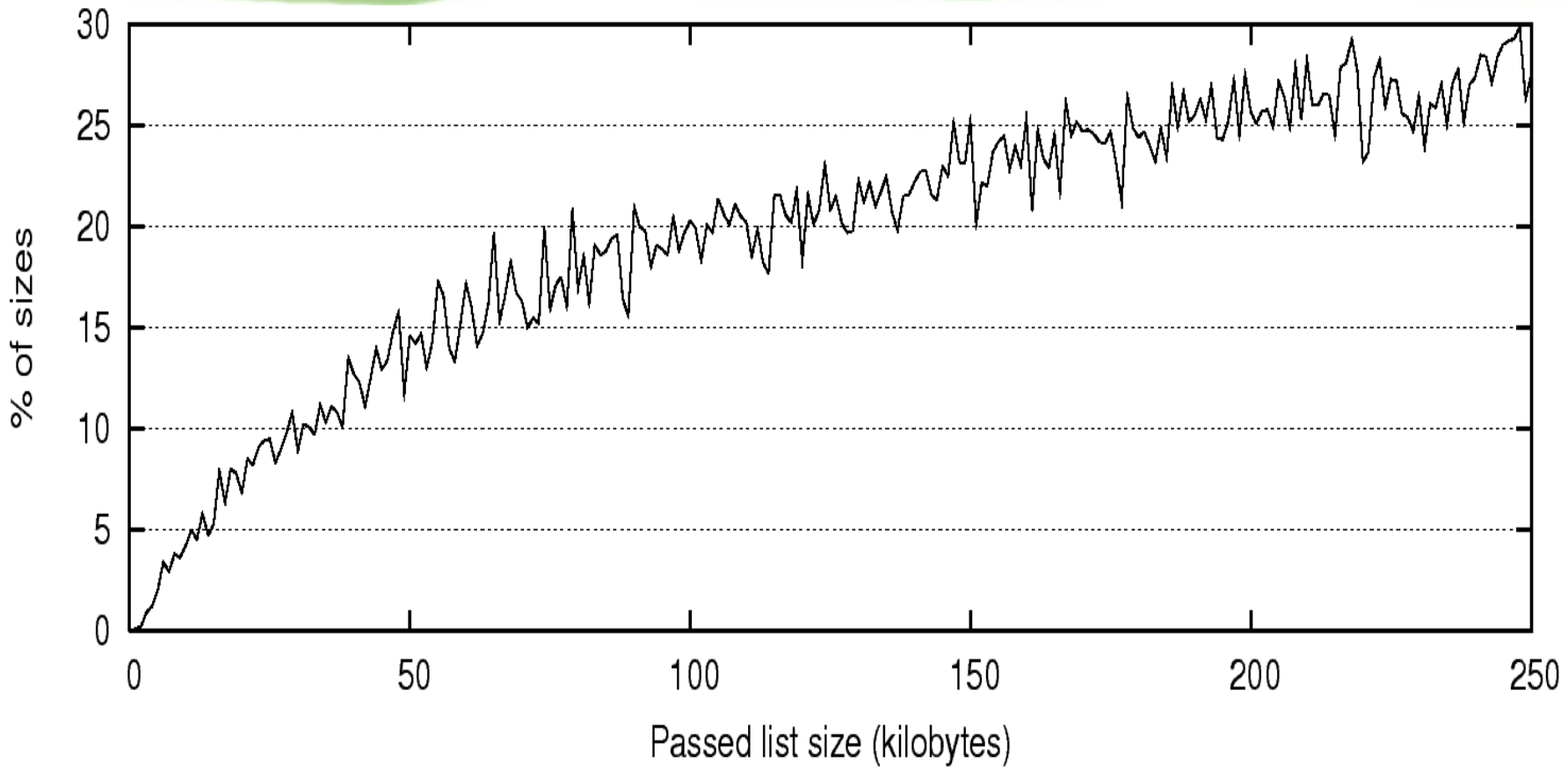
Hash functions 2

- Hash functions are well researched to be as pseudorandom as possible.
- Can we do better?
- Can we encode some relevant simple abstraction function into the hash function?

Hash table size sweep

- Start with a really small hash table size and modify the size of the table (the base of the mod function).
- Works well for synthesis tasks
  - task failed with exceeding 3 GB of mem in the explicit case;
  - worked with 100 MB of memory with bit state hashing enabled,
  - but

Hash table size sweep



- Percentage of queries yielding a trace to the desired state (not "may be").

Hardware vs software checking

- Hardware in general has a lot of control states and relatively few data variables
- Software has loooots of data and weird constructs like threads, dynamic creation of objects, garbage collection ...
- One has to be really careful when one wants to make bit-state caching work in a more general case.

Ideas

- By modifying the size of the hash table we got an answer to the query in seconds and by using a few kilobytes for the hash table.
- The cache memory of modern processors is 1-2 MB. This should make such sweep really fast.
- Processors with multiple cores are already available for laptops.

References

- Dillinger, P.C., Manolios, P.: Bloom filters in probabilistic verification. In Hu, A.J., Martin, A.K., eds.: FMCAD. Volume 3312 of Lecture Notes in Computer Science., Springer (2004) 367-381
- Ernits, J.: Memory arbiter synthesis and verification for a radar memory interface card. Nordic Journal of Computing 12(2) (2005) 68-88
- Holzmann, G.J.: An analysis of bitstate hashing. Form. Methods Syst. Des. 13(3) (1998) 289-307
- Zhang, L., Malik, S.: Cache performance of sat solvers: a case study for efficient 10. implementation of algorithms. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of Lecture Notes in Computer Science., Springer (2003) 287-298