Murf: A Retrospective

Prof. David L. Dill
Department of Computer Science
Stanford University
http://www.verifiedvoting.org

Murf

- Freeware system produced by our research group.
- Explicit-state model checker for simple safety properties.
- Widely used for cache coherence problems
  - Cryptographic protocols
  - User interface
  - Etc.
- Innovations
  - Practical symmetry reduction
  - State hashing
  - Parallel implementation
- Currently available: From University of Utah (Ganesh Gopalakrishnan research group).

Prehistory

Result of interviewing hardware people in my building (E.g., John Hennessy, Mark Horowitz, Anoop Gupta)

Q: “What is the biggest practical problem in hardware verification”

A: “Multiprocessor cache coherence”

(Not surprising, since these people were all working on the DARPA-funded DASH project – a shared-memory multiprocessor.)

Goal: Experiment with BDD-based Model Checking of DASH

Ken McMillan's work on "Gigamax" protocol was inspirational.

Three PhD students: Andreas Drexler, Alan J. Hu, C. Han Yang

Initial approach: Minimize language design; try ideas with minimum tool building.

Midway through the summer: Student opinion – building tools and doing the project would be faster than doing the project without the tools.

“Trans”

- Language was inspired by Misra & Chandy UNITY concurrency model
  - Iterated guarded commands
  - Asynchronous concurrency
  - Communication/synchronization through shared global variables
  - Parallel composition = union of rules.
- Trans was to be translated to BDD relations

Murϕ

- In one week, I saw two other languages/systems called "trans"
- Considered calling it Murphy (my version of Murphy's law: "The bug is always in the case you didn't test.")
- Saw another system named Murphy that week...
- Hence "Murf" – a name so silly that no one else would use it. (An early review: "Lose the cute name...")
- Subconsciously inspired by TeX?
- Lesson: Google specificity is a good thing.

BDD Verifier

- Alan Hu was the BDD specialist (victim)?
- Andreas Drexler wrote the front end (parser, etc.)
- Han Yang worked on applications.

Result: BDDs were a bottleneck.

Solution: Build a simple explicit-state verifier so we could develop some protocol descriptions -- while Alan got the BDDs working better.

Scaling up and scaling down

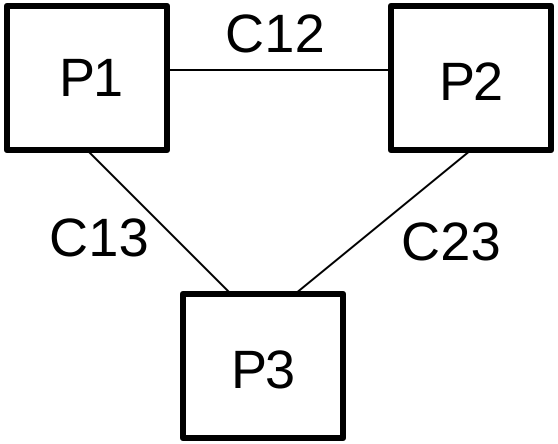
- Murphi's primary use mode was bug-hunting
- Only small descriptions could be handled
- *Downscaling Hypothesis*: many bugs can be discovered in a scaled-down system: 3 processors instead of 64, 2 memory addresses, 1 bit of memory.

Features:

- Symbolic constants
- "Rulesets" – parameterized families of guarded commands provide for scalable concurrency.
- Arrays – provide for scalable state variables.

BDDs

- Meanwhile, Alan was still trying to make BDDs work. Back end tool called “EVER”
- Problem: Multiprocessor cache coherence protocols are not good for BDDs.



P’s and C’s have BDD variables.

Typically, P_i sends a msg to P_j via C_{ij}.

P_i is in “wait” state, P_j in “receiving state”, C_{ij} has msg from i to j

All variables are correlated, so they should be adjacent in order.

But not all P’s and C’s can be next to each other.

BDD tricks

- Alan and I tried heroic measures to make BDDs work
  - “Functional dependencies” – identify BDD variables that could be replaced by functions of other variables.
  - “Implicit Conjunctions” – Represent the state space as the conjunction of a set of BDDs.
- Result: Papers were written, Alan got a PhD, and Murf continued being an explicit-state verifier.

Andreas Nowatzyk at Sun

- Computer architect – worked on Deep Thought at CMU while I was there
- Shared an office with Michael Browne, another student of Ed Clarke's
- At CMU, seemed to derive malicious pleasure from debunking the effectiveness of model checking.
- Thoroughly understood verification and wanted help applying it to the machine he was designing at Sun.

Sun's cache coherency

- I decided to try our new tool on an abstracted version of Andreas's protocol.
- This generated *many* requests for changes and improvements in the tool and language.
- After lots of work and several false positives, we started discovering protocol problems of interest to Andreas.

Lesson: If you want your group to produce a decent tool, use it yourself!

Symmetry reduction

- Our cache coherence protocols had huge amounts of symmetry
  - Identical processors, memory addresses, memory values, etc.
  - Many redundant states were being explored (swap processors 1 & 2).
- Idea: Symmetric subrange – elements are interchangeable (called “scalarset”)
- Proc: 1..n => Proc: ScalarSet{n}
- Operations on scalarsets must respect symmetry: Only assignment, equality, array indexing.
- Only captures full symmetry – but that’s the most important practical case.

Symmetry

- PhD student Norris Ip figured out the details:
  - Scalarset values represented as small integers.
  - If the values in the scalarset were permuted, resulting state graph is bisimular to original
- Implementation
  - State is not searched if an equivalent state is found in state table.
  - Starts are normalized before looking up in hash table.
  - Problem is equivalent to graph isomorphism, so heuristics were used.

Probabilistic search

- Visiting student Uli Stern
- Hash compaction (based on Wolper & Leroy's paper)
  - Store make hash table index, hash signature independent.
  - Use linear hashing
  - Use breadth-first search to minimize path length.
- Use of disk for state table (breadth-first search)
- Parallel Murf

Lessons

- Find a challenge problem to solve (e.g., DASH cache coherence).
  - Find motivated people to work with.
  - Solving it should be important.
  - Do things that work – then generalize.
  - Tool designers/builders should also use the tools.
- Benefits
  - Helps set priorities ("Is this helping to solve the problem?")
  - Tool will work for at least one thing (it's easy to make tools that work for zero things).

Lessons

- Minimize barriers to use
  - Liberal licensing (beware of corporate lawyers)
  - Minimize cost of "trying it out".
  - Don't make users download too much other stuff.
  - Make build/install easy.
- Work in Silicon Valley (or Austin, or …) where finding interested industrial people is relatively easy.
  - Early users/partners are crucial.
  - Later, tool sold itself.

Participants

- Alan J. Hu
- Andreas Drexler
- C. Han Yang
- Ralph Melton
- Norris Ip
- Seungjoon Park
- Ulrich Stern