

Maximizing the Value of Your Formal Run

Scott Meeth

scott.meeth@sbcglobal.net

Sun Microsystems, Inc

George Plouffe

george.plouffe@sun.com

Sun Microsystems, Inc

DAC 2010 User Track

Presentation Outline

- Ideal formal verification flow
- Fallback
 - > Making sense of bounded proofs
 - > Evaluating bounded proofs
 - > Case Study
 - > Ideas

Formal verification

- Functional properties
 - > Specify target properties
 - > basically, assertions about outputs
 - > Specify input constraints
 - > basically, assertions about inputs
- Goal
 - > Prove that the properties hold for all input sequences that satisfy the constraints

Formal Verification Success

- Success is relative
 - > To the quality of your properties
 - > To the quality of your constraints
- Ideal success
 - > Get confidence in your properties by using a model checker to obtain full proofs of them
 - > I ignore the question of the *strength* of the properties...
 - > Get confidence in your constraints in some robust way
 - > Probably the best way is *Assume/Guarantee*: prove that the logic driving your inputs satisfies the respective constraints

Life is like...

- Try for the ideal
 - > Hopefully you achieve it, and on schedule
- Often you have tradeoffs
 - > Invest more time in the effort
 - > To get better quality results
- If you have to settle for less-than-ideal outcome
 - > Want to maximize outcome quality vs invested effort
 - > Want to objectively assess the outcome quality
 - > You can often get a nice consolation prize

Outcome quality vs invested effort

- For any number of reasons, you may be unable to achieve the ideal outcome
 - > fully realized properties and constraints
- Suggestions, then, for next-best confidence in
 - > constraints (in lieu of Assume-Guarantee)
 - > properties (in lieu of convergence)
- Rest of talk
 - > assume we're resigned to settling
 - > no convergence for one or more targets
 - > no assume guarantee on inputs

Running in simulation environments

- Often a project has a well-developed simulation environment
 - > Thorough suite of directed tests
 - > Comprehensive constrained random environment
- Then you can get a fair amount of confidence in your constraints pretty much for free
 - > You really should run your constraints against simulation environments anyway
 - > Include them, and your properties, in the regressions if possible

Running in sim envs

- At the very least, this keeps formal and sim in sync
- If a constraint fires in sim
 - > Most likely your formal environment is over-constrained
- If a property fires in sim, and the property does not get a counterexample in formal runs
 - > If the formal runs converge, this is due to over-constraint
 - > sim env exercises input sequence your formal env precludes
 - so there's a constraint firing in there too
 - > If your formal runs don't converge, firing could be due to
 - > sequential depth: required input sequence is too long, or
 - > again, over-constraint is possible

Simulation environment

- Everything is relative
 - > If your project's simulation environment is robust and comprehensive, and people have confidence in it
 - > Then you can leverage this effort, and get a similar amount of confidence in your constraints (and properties) nearly for free
- Running the properties in formal figures to give you additional coverage
 - > Simulation and formal environments present stimulus differently

Simulation, formal complementary

- Formal and simulation: input sequence generation
 - > The sim environment: kind of depth-first search
 - > each test is a single input sequence
 - > breadth is obtained by additional runs
 - but how much additional breadth for how many additional runs?
 - > sequential depth is obtained by running a longer diag, which is linear in time/space
 - but how additional coverage attained with longer runs?
 - > The formal environment: kind of breadth-first search
 - > a run is like a constrained random suite, but with complete coverage (allowed by constraints, and perhaps up to a bound)
 - > but sequential depth is obtained at exponential cost in time/space

Bounded proofs

- Trying to get convergence: tools
 - > automatic:
 - > abstraction, design space tunneling, engine choices
 - > manual:
 - > design space tunneling, state space tunneling, abstraction
- If you need to make do, try to maximize depth of bounded proof
 - > And try to quantify the progress
- Bounded proof of N iterations means there is no counterexample of length $\leq N$ satisfying constraints

Case study

- Design
 - > Coherence Unit of processor
 - > Interfaces to L2\$, MCU, Link
- Model Checking tool used JasperGold
- Property
 - > When a Transaction ID (TID) is re-used, everything that needs to be finished from the previous usage is done
 - > What needs to be finished for given TID?
 - > depends on request type, where memory data is coming from, if cache data is coming, node configuration, etc

Maximizing depth of bounded proofs

- Combinations of engines (
 - > We happen to have access to lots of compute resources
 - > `set_engine_mode {h h d i b k}`
- Final deepest bound attained by B, H, K
 - > Occasionally engines *I, D* attained deepest
 - > It varied as we made changes to constraints
- Counterexamples from B, H
- Compute resource capacity, memory and runtime

What to make of bounded proof

- Quality of outcome is relative to
 - > Quality of counterexamples obtained along the way
 - > Depth of these counterexamples
 - > formal counterexamples are **much** more compact than sim; (nearly) every cycle is meaningful/necessary
 - > formal has tight control for generating mischief (corner cases)
 - > Depth the bounded proof reached
 - > Presence/absence of certain design constructs
 - > counters, FIFOs
 - > If you've not abstracted them, you need proof bound deep enough to exercise significant range of behavior
 - > In our case: no counters, shallow FIFOs (depth 1 or 2)

What we made of bounded proof

- Along the way, we got a 25-cycle counterexample
 - > Included the start and finish of a request,
 - > Followed by a spurious message from the design on behalf of the request, **after** it was cleared;
 - > Seeing this, when we got bounded proof of that property to 30 cycles, we felt reasonably confident (although...)
- Later, we got a 37-cycle counterexample
 - > Included start and finish of 2 requests consecutively
 - > Counterexample due to state from first request persisting into the second
 - > Constraint problem, but felt good about coverage depth

Outcome

- Properties
 1. 39 iterations: Engine H
 2. 81 iterations: Engine K (this one had nice 37-cycle cex)
 3. 129 iterations: Engine K
 4. 31 iterations: Engine B
 5. 31 iterations: Engine B
- Properties 1,2,3: locally sourced
- Properties 4,5: remotely sourced
 - not sure why they bogged down more than first three

Summary

- Primary goal:
 - > convergent properties
 - > verified constraints (Assume-Guarantee)
- Apart from that (for whatever reason)
 - > You can get a decent amount of confidence
 - > by putting Jasper to work to help give confidence
 - > by leveraging simulation environments
- Big caveat though
 - > Bounded proof of liveness property?
 - > Need some effort here to interpret; not so clear how useful