

Integrating Formal Into Main-Stream Verification: The IBM Experience

Jason Baumgartner, Viresh Paruthi
IBM Corporation

Thanks to: Hari Mony, Wolfgang Roesner

March 21, 2007

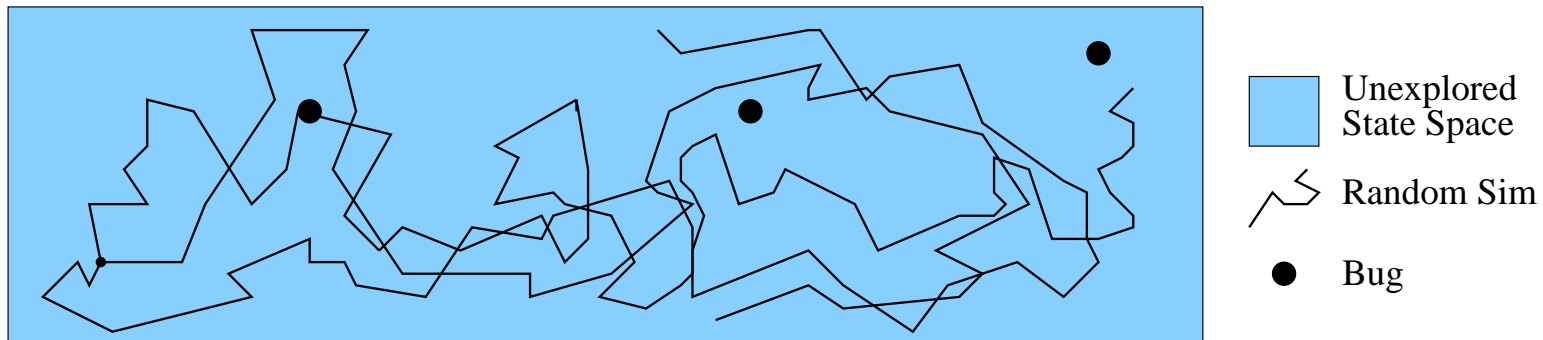
<http://www.research.ibm.com/sixthsense>

Overview

- Simulation vs. Formal Verification (FV)
- Bringing FV to the Masses
 - Fit FV within Existing Design Methodology
 - Enable Non-Experts to Leverage FV
 - Scale FV to Large Testbenches
 - Increase Return on Investment through Testbench Reuse
- Reusing Sim Testbenches in FV

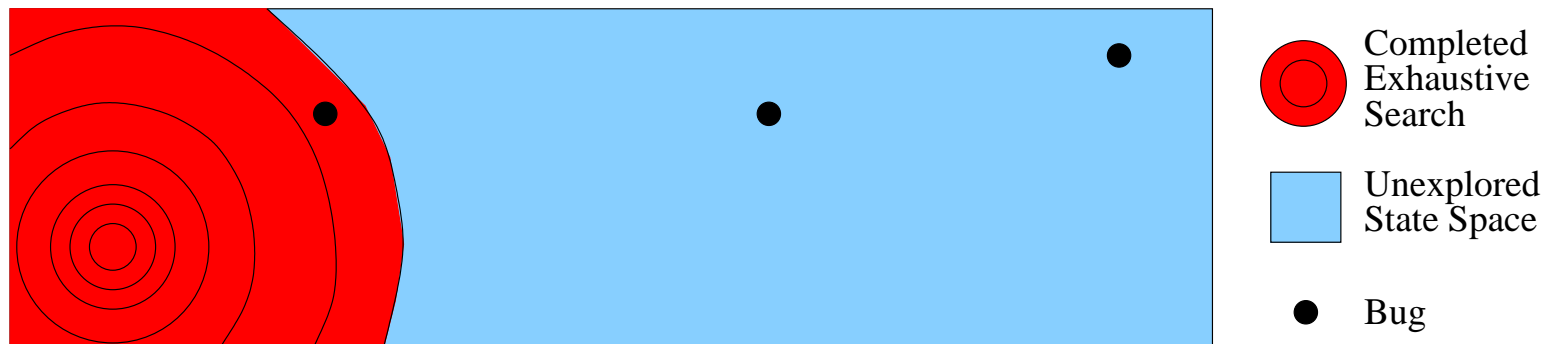
Simulation

- Validates the design against specific sequences of input stimuli
 - Scalable, though non-exhaustive: suffers the *coverage problem*
- Sim specs can be written using a variety of languages
 - + *Synthesizable* languages: PSL, SVA, HDL-based
 - *Non-synthesizable* languages: C / C++ variants
 - These languages cannot readily be reused in formal, emulation



Formal Verification

- + *Exhaustive* (unlike sim): finds corner-case bugs, yields proofs
- + Automated: easy to use, for smaller problems (block-level)
 - Substantial expertise, manual effort required for larger designs
 - More difficult to cover (micro-)architectural properties
 - A different type of *coverage problem*
- Requires synthesizable languages: PSL, SVA, HDL-based



Simulation vs. FV

- Sim retains predominant industrial framework due to
 1. *Scalability*: useful for tasks too large for FV
 - May refer to as *ease of use*
 2. *Risk* that formal spec may not pay off; merely choke FV tool
 3. *Legacy*: tools, skills, methodology using sim are well-established
 4. *Reuse of verif IP*: cost to **rewrite** sim specs in a formal language
- Though sim has its own drawbacks
 - Misses bugs!
 - Methodologies for high coverage are time-consuming

How can we close the Sim → FV Gap?

- *Full FV* of complex designs requires expensive, risky paradigm shift
 - A good goal, but needs to be eased into...

How can we ease into Wider-Spread FV? (1)

- Do not *require* a radical change in design paradigm to *enable* FV
 - Design methodology change has associated cost, risk in itself
 - Need for reuse of IP, skills, tools, methodologies is a high barrier
- While such a change may have many long-term benefits...
 - **There are many bugs to be found in today's design paradigm!**

How can we ease into Wider-Spread FV? (2)

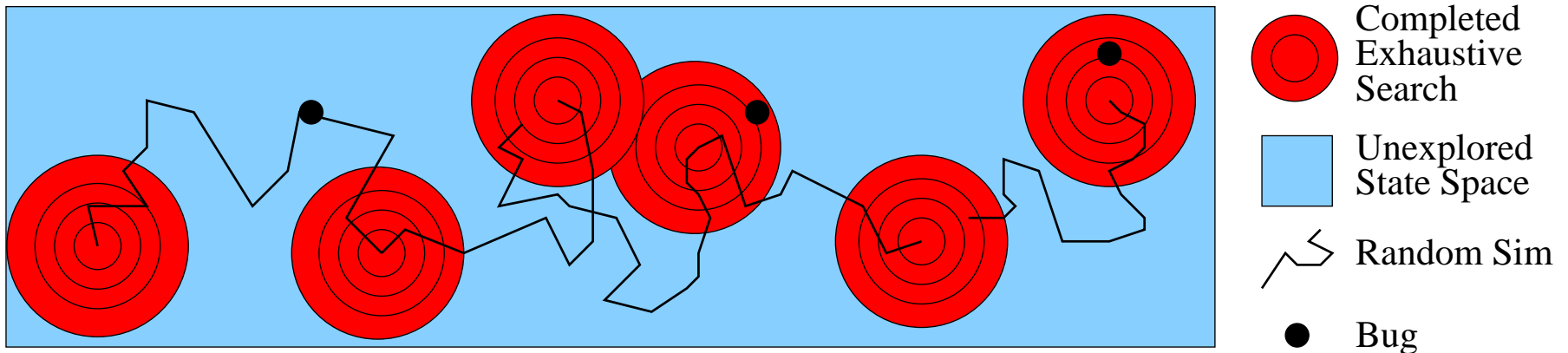
- Enable non-experts to leverage FV
 - Cannot expect verif+design team to all have PhDs in FV!
 - * Goal: make FV as easy to use as sim
 - * *Ease of use* requires **scalability and automation**
 - Costly to always throw learning curve of new design at FV gurus
 - * More cost effective for designer / *local verif team* to write specs?
 - Leverage easy-to-use sequential equiv checking paradigms

How do we achieve Scalability and Automation?

1. Tune system for importing and manipulating LARGE designs
2. Integrate falsification as well as proof threads
 - *Semi-formal falsification* improves ROI of formal spec
3. Integrate a variety of algorithms
 - Every problem is different
 - Different proof algorithms have different strengths / weaknesses
 - *Technological advances continue to push the capacity of FV*

Semi-Formal Verification

- Uses resource-bounded formal search to *amplify* simulation
 - Leverages simulation to reach *deep* states
 - Formal search triggered from deep states
- **Much more scalable** than pure formal; lessens formal spec risk
 - Very useful for quickly flushing out complex design bugs
 - *Enabling technology* for wider-spread formal



How can we ease into Wider-Spread FV? (3)

- Offer tangible *return on investment (ROI)* and *resource savings*
 - Scalability reduces negative ROI risk of formal spec development
 - *Leverage FV without substantial head-count increase*
 - * Cannot afford disjoint sim + FV team for every design component
 - Goal: reuse specs across formal + sim
 - Need to disperse FV spec and deployment from team of gurus

Testbench Reuse

- Requires scaling FV to unit-level testbenches
 - + More meaningful than block-level testbenches
 - + Better-documented interfaces to *drive*
 - + More encompassing properties to *check*
 - + **Verify functionality vs. verify blocks**
 - + More cost-effective: fewer testbenches to *cover* design
- Big, ugly testbenches may need tweaking for optimal formal results
 - Reserve FV gurus for this purpose
 - (and for emergencies)

Conclusion

- IBM SixthSense philosophy: *non-intrusive FV*
 - Scale FV to sim-sized testbenches
 - * Integrate semi-formal, and variety of synergistic algos
 - Ensure high automation, ease of use
- Push for reusable testbenches across sim + FV
 - Greater ROI of specification investment
 - Disperse formal spec effort; retain FV gurus for critical tasks
- Result: substantially wider-spread use of FV