Formal Verification As a Technology Transfer Problem

Robert Kurshan
Cadence Design Systems

In 1983 I used the first version of the COSPAN model checker (written in Fortran!) to verify a part of a LAN link level protocol that was being implemented inside AT&T. Although the model checker was very rudimentary, it found 11 bugs in the design specification, 3 of which had made it into the implementation. Finding those 3 bugs launched my career in model checking. Having found them early in the design process saved enough redevelopment time that the project manager wrote a nice letter to my management and offered to pay for my continued debugging services.

Three years earlier, Ed Clarke implemented his EMC CTL model checker, presenting a decidable logic that turned out to have great practical value to the development community, especially for hardware.

Even earlier, in the late '70s, Colin West at IBM showed how one could systematically explore the state space of a protocol model and use the exploration to find bugs in the model.

In the ensuing 25 years, the sophistication and power of algorithms and methodologies for finite state verification decision procedures grew steadily, and Computer Aided Verification evolved into a recognized discipline in Computer Science. By the late '80s, this work was beginning to see application to products, especially in hardware. Sophistication and application continued to grow through the '90s.

Around 1990, various groups began to sense that the methods were sufficiently mature to warrant turning them into commercial tools. With ever-increasing design complexity, the cost of design testing was consuming an ever-greater portion of the design budget – as much as 80% – so more powerful methods for testing became badly needed. However, it soon became evident that what was useful in the hands of experts could not be directly converted into a useful commercial tool.

Why was this? In retrospect, the reasons are clear and obvious:

1. they required a significant change in design methodology;
2. developing the verification algorithms into a commercial-quality tool was expensive;
3. given 1., the market for these tools was hard to estimate.

The required methodology change included advancing verification to earlier stages of the development flow. This offered a big potential advantage for reducing development costs, since finding bugs earlier could save more costly fixes later. However, it required that developers participate in the verification process (at least to the extent of writing properties to be checked). There was and still is considerable reluctance to do this:

1. it is viewed as a diversion of highly-valued developer resources into what is broadly considered a more menial task of testing;
2. especially some years ago, but still today, there is concern regarding the usability of the tools and a reluctance to allow the highly-valued developer to be a guinea pig for a new technology;

3. any methodology change requires some planning, and the ubiquitous gaspingly tight development schedules often do not offer the extra resources needed to do this, especially at the level of the developers who drive the project and hence its delivery schedule.

An alternative, to have a special verification team write properties and then check them, would not work in a scalable fashion. Such a verification team would be too far removed from the design to know all the properties that need be checked within blocks. Extracting that information from the designers would be unacceptably disruptive to the designers (being about the same as asking them to write the properties themselves). And writing only global properties as done in system test would lead to capacity issues that were not easily overcome by non-experts.

Meanwhile, proof-of-concept demonstrations of the value of formal verification, at least in the hands of experts, become ever more persuasive, while the urgency for a better solution to design test grew rapidly. This emboldened quite a few tool vendors in the Electronics Design Automation (EDA) industry in the late '90s to project an adequate return on investment to initiate the development of commercial tools with the prospect of broad sales in the near future of their deployment. The number of EDA vendors in this initial round heightened expectations and thus the level of disappointment when all did not work out as hoped. All of the early entries failed, and with their failure came a reassessment of the value of formal verification.

In a typical fashion, hyper-inflated commercial expectations led to hyper-deflated despair when all did not work out as planned. It became common to hear that “formal verification does not work”. The reasons given largely focused on the technical truism that the underlying algorithms do not scale to adequately large problems. The fact that for years, the algorithms had been producing impressive results in the hands of experts was dismissed with the sense that there were insufficient experts around to support sufficiently broad utilization of the tools to render them profitable for the EDA vendors.

Nonetheless, the urgency for a better solution to the testing problem continued to grow, and with this grew the utilization of the same tools that had been deemed inadequate. The EDA industry took notice and a second round of competition began.

In this second round, it was clear to the industry that the key to success was not so much intrinsic power as usability. While intrinsic power, measured in terms of performance and capacity (the size of designs that could be handled) was critical, the gating issue was now seen to be how easy it would be to break into the current development flow with the new tools. Thus, for initial market penetration, the gating issue was not performance and capacity, but usability. Performance and capacity would quickly become gating issues in the following phase, once the adoption barrier was breached, but the initial penetration problem was finally clearly seen to be a technology transfer problem.

To be successful, technology transfer often needs to be accomplished through a succession of small steps, each of which is barely disruptive to the status quo, but nonetheless shows some measurable benefit. The lack of disruption is necessary to overcome
the reasons for resistance to this (or any) new technology. The measurable benefit is needed to justify the next small step.

The succession of small steps is sometimes illustrated with an "adoption curve" that plots time against number of users or tool revenues. An important accompanying curve plots time against tool development cost. Superimposing the two curves, the adoption curve must cross the cost curve sufficiently soon and with sufficiently higher slope to warrant the EDA project start up costs, as weighed against the projected return on investment.

Thus, solving the technology transfer problem is tantamount to finding a suitable adoption curve, that is, finding a succession of small incremental adoption steps that will overcome the adoption barriers.

In hardware testing, an important catalyst for a solution lay with the standardization of a query language - the means for defining properties to be checked. With the adoption of PSL as an industry standard query language, resistance to writing properties into the design diminished. Moreover, since properties expressed in PSL could be used not only for formal verification, but also as monitors in simulation test, it became much easier to justify their adoption. Designers, without being diverted to testing, could be asked to write their design comments in PSL instead of English, enabling a verification team to use them in both testing and formal verification without the need to learn the details of the design. The fact that these PSL properties could be used as simulation monitors meant that the incremental cost in terms of methodology change for formal verification was very small. Thus, the adoption of PSL provided the first point on the adoption curve.

With PSL properties in place, formal tools could be configured to automatically analyze arbitrarily large designs, picking off properties one at a time to check. Each check could automatically be localized to the relevant part of the design, and automatic assume-guarantee reasoning could be employed to use some properties as assumptions while checking other properties. (This ameliorates the problem of defining the environment - constraints on inputs - when a block is checked in isolation for a local property.) This would be a second small incremental step in the adoption curve.

Successive steps include manual intervention for better results in support of verifying less-local properties, verifying properties when there are insufficient properties defined to support automatic assume-guarantee reasoning, and for user-guided abstraction.

New algorithms are deployed to reduce or eliminate the prospect of "false" errors - errors found in an abstraction that would not be possible in the actual complete design. (There is a tradeoff in measuring the loss from discarding an actual design bug that cannot be automatically verified as such, against the user cost of analyzing and then discarding potential bugs that turn out to be artifacts of the abstraction process. While some tools adopt an extreme strategy of discarding all bugs that are not automatically verifiable, a more cost-effective strategy may be to rank potential bugs and present those that are most likely to be actual design bugs.)

With the view of formal verification as a technology transfer problem, a succession of small evolutionary steps can be mapped, each of which causes small methodology disruption, but leads to some positive benefit. A commercially viable succession is one
whose adoption curve when measured against its investment cost curve, shows adequate return on investment.

Software verification today is about where hardware verification was a decade ago. The problem is intrinsically more difficult and thus will be the commercialization of its solution. However, the technology and proof of concept are in place, and it is a good time to begin plotting an adoption curve for commercial tools in support of formal verification of software.