

Toward Formal-Methods Oecumenism?

Fabrice Kordon, *Université Paris 13 & Marie Curie*
Laure Petrucci, *Université Paris 13*

Arguing that there's only one way to do formal methods is common practice, when in fact interoperability between formal methods can and should be achieved.

The scene takes place in a pub (probably after a symposium on formal methods!). Three researchers are having a discussion over beers in a corner of the room. They don't pay much attention to the music or any other distractions—the discussion is animated. Let's take a closer look. Are they fighting about Roland Garros, Wimbledon, or the results of a World Cup soccer game? No. The topic seems to be formal methods.

An animated discussion

ALICE : I can't let you say that model checking is a way to prove programs! Mathematics are mathematics; you can't consider model checking to be that related to mathematics since your verification isn't parameterized at all! Model checking is a test, not a proof.

BOB : But graphs are mathematics. Claude Berge is considered a famous mathematician. The state space generated by a program using model-checking tools is just a way to exploit such a structure.

CHARLIE : Sorry, Bob, Alice is right. When you prove your system using model-checking techniques, you just consider one configuration. Actual mathematics are more than that. When you use model checking to prove your system, it might be impossible to evaluate the impact of the initial configuration on it.

BOB : At least we might prove something easily. Can you imagine what a programmer would do with your theorems and axioms?

ALICE : You're not being fair. There are lots of stories of theorem provers—or equivalent techniques such as B, Z, or other algebraic methods—solving problems.

BOB : Yes, but everybody can set up a Promela program, and run SPIN on it, and it works without the help of those smart guys from great universities. There are too few of them, and automated tools are necessary to conduct these proofs.

CHARLIE : Theorem provers are here to help and to make these mathematics easy for everyone. Plus, you also have an environment—for example, in Coq. They bridge the gap between the semantics of programming languages and theorem provers.

BOB: Come on! They only enable functional programming. Who actually uses functional programming in real distributed systems? Take a closer look—most programs are in C, Java, or C#, and model checking starts to be operational for such programs. Transparency is the key issue, and it's a way to hide the complexity of the underlying formal methods.

CHARLIE : On this point, I agree, but with Petri nets, you can also enjoy the benefits of structural analysis without deploying the system's state space. So, you can, by means of invariants, parameterize your property according to the initial system state. Then you also have the possibility of doing model checking, so you get the best of both worlds.

There is a big smile on Charlie's face. Bob's eyes are furious. Alice drinks some beer and then shows a triumphant expression.

ALICE : You guys are wrong: model checking is just an interesting testing technique. Let's assume it's more efficient but only considers finite systems. And your Petri net stuff, Charlie, is a pleasant farce. I agree that there is parameterization in your invariants, but how poor are they when you want to analyze and verify a given property? Petri nets are too poor for the elaborate properties of complex systems.

BOB : Such a miserable argument upsets me. Efficient techniques now exist for addressing combinatorial explosion in model checking. Moreover, how valuable is an unused theorem? It's as useless as a dead programming language. I agree that for some points, theorem proving is important, but when you must build rapidly distributed systems, all weapons are important, and the challenges and techniques used in model checking are as formal as any other.

And the argument continues...

What's so important about formal methods?

Hearing discussions that use unfair arguments to promote one's work is quite common. But, despite differences in the techniques, every formal method has its strengths and weaknesses.

For example, as Bob says, model checking is easy to use (and to implement, in a naive way). Of course, when dealing with very large systems or with specific structures to handle complexity, implementation is much less trivial—but it remains user-friendly for engineers. And that's a good point. Moreover, it's very relevant for parallel systems—another good point.

However, model checking is rather poor when properties deal not with causality of events but with data types or recursive constructs. In that area, algebraic methods, even if they're more complex to grasp, are much more appropriate and efficient.

And it's true that, with Petri nets, you can do both model checking and a sort of parameterized analysis. But, if model checking remains as efficient as what you can get from a Promela program, the model itself only offers basic constructs to handle parallelism. This is also true with parameterized proofs using Petri net invariants: it's possible, but only on a set of predefined properties that still have to be interpreted. This is much less than what you can do with algebraic-based methods, but there is a systematic algorithm to do so. Furthermore, it doesn't require a struggle to find a path from axioms to the property to be verified.

But everybody is used to arguing that there's only one way to do formal methods.

Like Martin Luther King Jr., "we have a dream." We have a dream in which people recognize all methods' strengths and weaknesses—and all methods can deal with each other and join forces to combine their strengths. Fortunately, we see signs that this time is coming. For example, both the software engineering and hardware industries are becoming more interested in formal methods and are supporting joint projects with academics. Maybe a challenge such as introducing formal semantics into UML and verifying UML specifications requires cooperation from the entire formal-methods community, because in today's world, we need verification tools to enforce reliability in complex systems (regardless of the techniques involved).

Since enough problems exist on which at least one of the current techniques is efficient

(asynchronous versus synchronous systems, algebraic methods versus model checking, and so forth), we hope that the time of wars and clefts comes to an end and an era of peace and union is born, creating a bright future for formal methods.

Fabrice Kordon is a professor of computer science at the Université Pierre and Marie Curie. He's also head of the Modeling and Verification team in the Network and Distributed Systems department at LIP6 (Centre National de la Recherche Scientifique, Unité Mixte de Recherche 7606). Contact him at fabrice.kordon@lip6.fr.

Laure Petrucci is a professor of computer science at the Université Paris 13 and a member of the LIPN (Centre National de la Recherche Scientifique, Unité Mixte de Recherche 7030). Contact her at laure.petrucci@lipn.univ-paris13.fr.

Related Links

- DS Online's Software Engineering Community, <http://dsonline.computer.org/portal/site/dsonline>
- "A Panacea or Academic Poppycock: Formal Methods Revisited," Ninth IEEE International Symposium on High-Assurance Systems Engineering <http://doi.ieeecomputersociety.org/10.1109/HASE.2005.3>
- "Translating Diagrams: A New Approach to Introducing Formal Methods," 18th Conference on Software Engineering Education & Training, <http://doi.ieeecomputersociety.org/10.1109/CSEET.2005.39>

Cite this article: Fabrice Kordon and Laure Petrucci, "Toward Formal-Methods Oecumenism?" *IEEE Distributed Systems Online*, vol. 7, no. 7, 2006, art. no. 0607-o7002.