Veil

A Framework for Automated and Interactive Verification of Transition Systems

George Pîrlea

NUS National University of Singapore

joint work with Vladimir Gladshtein, Elad Kinsbruner, Qiyuan Zhao, and Ilya Sergey

Hello everyone. I am George Pîrlea and I am excited to share **Veil** with you today.

Veil is a framework for verifying transition systems that combines automated and interactive approaches in a single multimodal verification tool, embedded in Lean.

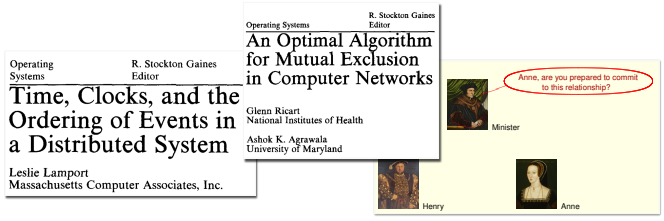
A Framework for Automated and Interactive Verification of **Distributed Protocols**

2

Whilst you can in principle verify arbitrary transition systems in Veil, it's really geared towards verifying distributed protocols.

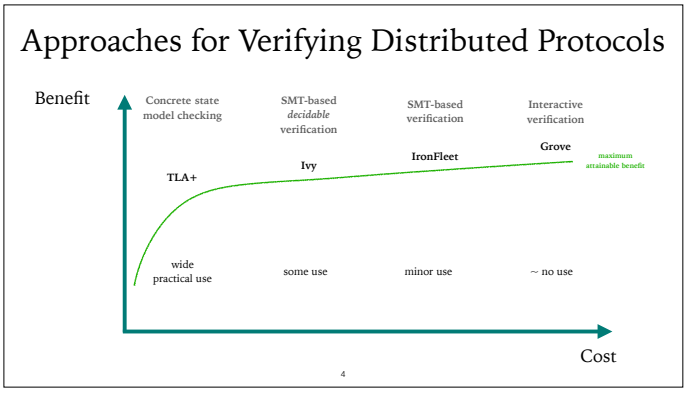
Distributed Protocols

Define how multiple parties collaborate with each other to achieve a common goal 🤝



3 *Two-Phase Commit* depiction from Leslie Lamport's video course on TLA+

A distributed protocol is a transition system that defines how multiple parties collaborate to achieve a common goal, usually by communicating over a network. There's all sorts of protocols out there, and it turns out they're quite tricky to reason about, so it's helpful to have tools to aid us in that reasoning.



There are many approaches to building assurance in these protocols, and we can plot these approaches on a chart, where the *x* axis is the cost or effort involved in applying these approaches, and the *y* axis is the benefit you can get from them, in terms of number of bugs found, types of properties you can verify, and so on.

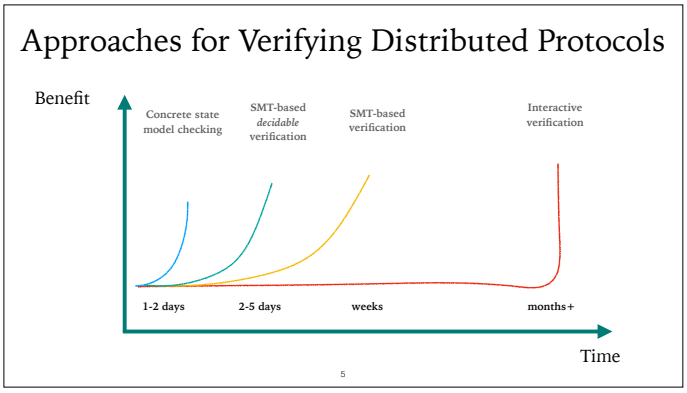
On the far left side, we have approaches that are low-cost, and chief among these is model checking, and in particular concrete state model checking using tools like TLA+ and TLC. These tools are mature and easy to use, so they are used extensively in industry, to build assurance in distributed protocols.

Taking more effort, tools like Ivy, require you to fit your protocol description in a decidable fragment of FOL, which requires more expertise, but let you actually verify — not just test — your protocol entirely automatically. These are also used in industry, but to a much a lesser extent.

And then there are approaches that are mostly academic, frameworks such as IronFleet and Grove, which use tools such as Dafny, Viper, or Rocq as the substrate for verifying distributed protocols. These let you prove very complex properties, but see relatively little use in industry.

And the question is: why? **Why are these more powerful approaches not used?**

Well, if you plot the maximum juice you can squeeze out of these approaches, so to speak, you get a curve that looks like this. A big initial jump from testing and model checking, and then a gradual slope upwards.



But it generally does not make sense to squeeze all the juice out of these respective approaches because they take immensely different amounts of time and effort. So here the X axis is time and Y axis is benefit, as before.

Modeling a protocol in TLA+ and checking it with TLC takes on the order of a couple days. Specifying a protocol in Ivy takes a bit more, but still on the order of a week. Using Dafny or Viper to verify a protocol generally will take longer, and using an interactive theorem prover will take longer still.

The problem is these very powerful approaches, as you can see, have huge upfront costs — you spend weeks, months, sometimes years before you start seeing any concrete benefit, and that's just impossible to justify in a practical setting.

So what people do in practice is they start with the approaches on the left, and more than 90% of the time, that's good enough. When it's not good enough, for instance, when you want to verify properties that are higher order or that the automation simply does not handle, then you have two potential options.

When Your Tool is Not Sufficient

You either:

- use a **combination** of tools, or

- add an **interactive escape hatch** to your automated tool

You either use a combination of tools, and that has well known issues and requires some duplication of effort, OR

You add an interactive escape hatch to your automated tool. And the user experience for that tends to be sub-par, to put it mildly.

My message in this talk is: there is another option. A better option.

Veil

It's called Veil.

It's not the verifier to kill all verifiers *yet*, but it does solve this issue.

And the way we did it is quite simple…

We just built the whole verifier in Lean!

We just built the whole verifier in Lean!

Veil

- ✅ A verifier and DSL shallowly-embedded in Lean
- ✅ Symbolic model checking via SMT
- ✅ Out-of-the-box interactive proofs in Lean
- ✅ Formalized meta-theory: sound VC generation

Lean is expressive enough that you can do all these things. So Veil:

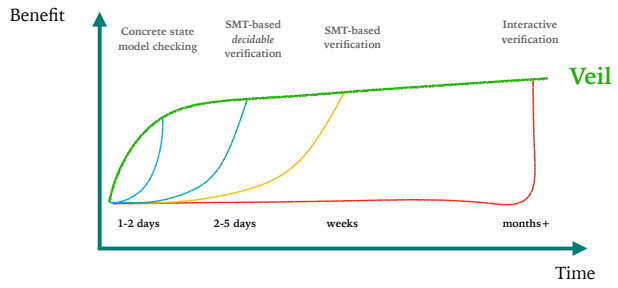
- Is a verifier built in Lean with a shallowly embedded DSL for specifying protocols;
- It does symbolic model checking by calling external SMT solvers — with proof reconstruction, if you want;
- When the SMT solver doesn't cooperate, you can just prove your VC manually — because every VC in Veil is just a regular Lean goal;
- And because this all embedded in Lean, you get to formalize and prove all the meta-theory you want. So we've proven that our VC generation is sound.

Demo

Here's what it looks like.

Points:
The way you'd use Veil is, everything that can be done automatically, you do automatically, and what cannot be done automatically, you do manually, WITHOUT having to switch to a different tool and WITHOUT having to rewrite your specification.

The Future of Verifying Distributed Protocols



11

It's not there yet, but the hope is eventually, with Veil, things will look like this.

A single expressive tool in Lean that seamlessly employs all of these approaches, so you only pay the cost imposed by the complexity of your problem rather than by the inadequacy of your tools.

Take Aways

- **Veil** is a Lean framework for *automated/ interactive* verification of distributed protocols
- *Shallow embedding* of the language, VCs are generated via a Dijkstra monad
- *Foundational*: different VC generators are proven sound wrt. each other
- Acceptable performance for FOL via SMT, *seamless integration* with HOL specifications

github.com/verse-lab/veil

Thank you!

12

Thank you very much. That's all I have.
