Leslie Lamport: Video course on TLA+ (lamport.azurewebsites.net)
328 points by kelvich 61 days ago | hide | past | web | 74 comments | favorite

ahelwer 61 days ago [-]

Great to see this here! I act as a TA for Dr. Lamport's TLA+ courses at Microsoft, and can answer any questions y'all have.

TLA+ in one sentence: it is a language used to write specifications, same as you might write a spec in English/your chosen informal language, except here you write your spec in basic mathematics; benefits of a formal specification language include freedom from ambiguity, model-checking, and even machine-checked proofs of correctness.

This language is a joy to use and I've found it really affects the way I think about system design.

djb_hackernews 61 days ago [-]

I haven't watched the videos yet but I have been keeping my eye on TLA+ ever since seeing the whitepaper from Amazon and how they use TLA+ to spec out their distributed systems. As someone that works on distributed systems that could use some formality I *think* TLA+ could help, however, I have a hard time really understanding it.

Do you know of any straight forward non-trivial examples? Something a little more complex than "hello world" and something that isn't abstract?

ahelwer 61 days ago [-]

There's a few I can think of! First, the TLA+ GitHub repo has a bunch of examples[0]. The best one to start with is the Die Hard puzzle[1], where the heroes must obtain four gallons of water given only three and five gallon jugs. In the spec[2] we set up the system and specify all possible actions; then we model-check the system to see whether it can spit out an execution trace of actions to reach a state where we have four gallons of water in a jug.

The Wikipedia article also has some good example specs[3].

For your own first specification, I personally cut my teeth on a river-crossing puzzle with a farmer, wolf, and sheep.

[0] https://github.com/tlaplus/Examples/tree/master/specificatio...

[1] https://www.youtube.com/watch?v=6cAbgAaEOVE

[2] https://github.com/tlaplus/Examples/blob/master/specificatio...

[3]https://en.wikipedia.org/wiki/TLA+

hwayne 61 days ago [-]

I wrote an essay[0] about a non-trivial example I did at my company. We had to work with several finicky APIs and the TLA+ spec caught several critical bugs with it. Plus, it's purely a business logic system, so it's pretty concrete.

If you're interested in learning more, I also wrote a beginner's guide[1] to the language, which contains concrete examples and exercises.

[0] https://medium.com/espark-engineering-blog/formal-methods-in...

[1] https://learntla.com/introduction/

chillitom 60 days ago [-]

Following your introduction now, it is a great quick-start tutorial, thanks for creating it.

jdubray 60 days ago [-]

I have created the SAM Pattern [1] (State/Action/Model) based on my interpretation of TLA+. The goal is to make the semantics available to developers. Why not writing code as close as possible to the way it would be specified?

SAM can also be used for stateful API/Microservice orchestrations [2]

If you want a slightly more formal introduction to SAM and its relationship to TLA+ (again, based on my own interpretation) I gave this lecture last month [3].

[1] http://sam.js.org

[2] http://www.ebpml.org/blog15/2015/06/designing-a-reliable-api...

[3] http://cloudsentinel.com/sam-state-machines-and-computation....

foxaal 59 days ago [-]

You wonder if there are code generation possibilities...

jdubray 57 days ago [-]

Actually, TLA+ (again the way I understand it), after nearly 20 years of MDSE, convinced me that I would very rarely need to generate code. It's a much better value proposition to write correctly factored code. TLA+ is one of the most amazing element of Computer Science. Everyone who writes code should have at least a basic understanding of it.

agentultra 60 days ago [-]

As someone who worked with distributed systems for many years before using TLA+ to find bugs and validate work: It's possible to understand it.

I think it would help to have a good mentor, as I did, or a lecture series from the language inventor. I ran through the trial version of these videos when Lamport was developing them and it's quite approachable even for someone who's more of a liberal arts kind of person.

colanderman 60 days ago [-]

I work on distributed systems and have started pushing for us to use TLA+/PlusCal to model them. I have a simple PlusCal model of a system with two threads which juggles network and disk I/O which I can e-mail you if you like. It's not at all a complex system, but I think it captures the basics of modeling concurrency and I/O well.

anateus 61 days ago [-]

Kind of an out of left field question: Lamport appears in a little square of video occasionally overlaid on the slides and sometimes full screen. As far as I can tell, he never recycles an outfit and seems to switch head gear regularly. The switch is often between segments that are quite short.

It feels almost like an intentional joke. Do you think that is the case? Or just even short segments were recorded on different days and Lamport like to wear all sorts of hats? That one where he's wearing a beanie and indoor sunglasses seems particularly intentional.

I've never met him, and the delivery of the lecture seems fairly dry and serious, so I can't tell :)

ahelwer 61 days ago [-]

Haha yep, it's intentional. He has a sense of humor like that. Famously, he first presented his paper on the Paxos algorithm dressed like Indiana Jones with the fictional backstory of the algorithm being an archeological discovery of the ancient parliamentary systems of the Greek island of Paxos.

AlphaSite 61 days ago [-]

The paper is more of the same: https://www.microsoft.com/en-us/research/wp-content/uploads/... addmitadly, it's also fairly difficult to follow the random interspersed bits of Greek.

gghh 60 days ago [-]

I have a passing familiarity with proof assistants such as Cow and HOL. How would you compare TLA+ with those?

https://en.m.wikipedia.org/wiki/Coq

https://en.m.wikipedia.org/wiki/HOL_(proof_assistant)

ahelwer 60 days ago [-]

Great question! So, TLA+ does have a machine-checked proof system called TLAPS, but its intended use is quite a bit different than Coq. While Coq is very well-suited for doing deep reasoning about mathematics and proving all sorts of stuff about whatever mathematical construct you care to define, TLAPS is focused much more on "mile wide, inch deep" type proofs required to demonstrate correctness of algorithms. Most algorithms use very simple mathematical properties, but have a bunch of edge cases and cyclomatic complexity which TLAPS is well-equipped to handle.

There might come a day where TLAPS is as powerful as Coq for writing general mathematical proofs (indeed, better ways of writing proofs and mathematics seem to be a passion of Dr. Lamport's, see his paper *How to Write a 21st Century Proof*[0]), but for now it's far, far easier to prove even simple statements about the Natural numbers in Coq.

[0] http://lamport.azurewebsites.net/pubs/proof.pdf

pron 60 days ago [-]

This is a complicated question. I am currently writing a blog post series on the *theory* of TLA+ that I will publish in May that, among other things, touches precisely on this point. But I will try to quickly point out some areas of difference, but the most important thing to remember is that both Coq and TLA+ -- *as far as specifying and verifying software is concerned* (not general math) -- are universal formalisms. Ultimately, anything you can do in TLA+ you can do in Coq and vice-versa. Also, when using mechanical proof to verify software, the proof techniques and the actual work is pretty much the same.

1. Difference in theory -- the theory of computation in Coq and HOL is based on functional programming; especially in Coq (I know very little about HOL), algorithms are functions in the FP sense (lambda expressions). In TLA, algorithms are state machines, and there's a richer notion of relationship between algorithms. This makes some things easier in TLA: Instead of programs being equal extensionally or definitionally, they can be equal or similar in many ways. In fact, in TLA there's a partial order relation on algorithms that reflects the refinement/abstraction relation. Also, it means that different kinds of algorithms -- sequential, concurrent, parallel, realtime etc., are all defined in the same way using the exact same ideas. This also makes it very easy to specify some properties -- such as worst-case complexity -- that are tricky in Coq. Having said that, people working on realtime systems in Coq implemented a simple version of TLA in Coq.

2. Differences in goals and audience -- Coq and HOL are ultimately general mathematical proof assistants. TLA+ is a tool for reasoning about digital systems. Coq and HOL can obviously reason about programs and TLA+ can prove general math theorems, but the focus is different. For example, Coq is used by logicians to explore new logics and mathematical foundations; you don't want to use TLA+ for that. On the other hand, TLA+ is used by "ordinary" engineers in industry working on "ordinary" software, while Coq and HOL are virtually unused in industry, and in the few cases they are, it's almost always in conjunction with academics and academic projects.

3. Differences in tools/abilities -- Coq can generate runnable code and be used for end-to-end verification. TLA+ cannot generate code, and while it has been used by academics for end-to-end verification, that is most certainly not its intended use. It is intended to be used in modeling large, complex systems at an abstraction level that's significantly higher than the code to find errors in design. As far as general math proofs are concerned, Coq is more capable than TLAPS, the TLA+ proof system. On the other hand, TLA+ has a model-checker that makes its proof checker almost irrelevant for large, complex specifications. As the users of TLA+ usually don't require a 100% watertight end-to-end specification, they're content to get the less than 100% guarantee of the model

checker in exchange for what is at least a one order of magnitude reduction in effort.

In short, I would say that tools like Coq have a more academic focus, and are in general more interesting to academics and less to engineers, while TLA+ has a more industrial focus, and is of more interest to engineers and less to academics.

But I will repeat and say that while there are big difference in theory, goals and usage, ultimately when working on verifying software, the challenge is in thinking about your system mathematically and understanding how and why it works (or not). This work is similar both in effort and technique no matter what formalism you choose to use.

agentultra 60 days ago [-]

I too have a series of blog posts on TLA+ and verification in general aimed at working programmers. I'm giving a talk next month at a local developers' meet up on formal specifications and will probably cover TLA+ mostly (or maybe Dafny?).

I'd love to read your posts. Is there a url or twitter account where I can get updates on your work?

pron 60 days ago [-]

I expect to publish the posts starting in mid-May. My twitter handle is pressron

unboxed_type 60 days ago [-]

AFAIK, TLAPS does not support temporal reasoning in full currently, so you are not able to prove liveness properties of your system. On the other hand, Coq is able to express both LTL logic and infinite trace, so you can prove such things in it.

pron 60 days ago [-]

TLAPS supports temporal reasoning. I've personally used it only for safety (inductive invariant), but to make sure it supports liveness, I just checked the following:

```
    THEOREM ASSUME NEW F, NEW G,
                   F ⤳ G, F
            PROVE ◇G
     PROOF BY PTL
```

and it works fine (PTL stands for propositional temporal logic). I suppose there are some things that aren't supported (TLAPS has quite a few missing features), but TLAPS is very new by proof-system standards. Note that in general, TLA+ tries to avoid temporal logic as much as possible. Lamport has repeatedly said "temporal logic is evil". A 1000-line spec, will probably have no more than a couple of uses of temporal operators.

(Of course, most people don't bother writing proofs at all for real, large, complex software as that is just too costly. They just use the model checker, that can check liveness, too.)

unboxed_type 60 days ago [-]

Temporal reasoning is a lot more than that you mentioned, so by providing this example you can not conclude that TLAPS support temporal reasoning. I have read several papers of S.Merz, Lamport's student and now a researcher, who mentioned that there is no temporal reasoning apparatus available in TLAPS. Maybe something have changed since then but I found no evidence for this.

> They just use the model checker, that can check

> liveness, too

This is not quite true. Liveness is about infinite number of states, but any real model checking is bounded by definition.

For an engineer this is not a problem at all, but as for comp.sci researcher your ability to use TLA+ is limited in this respect.

> Lamport has repeatedly said "temporal logic is evil"

Well I respect Mr.Lamport`s point, but to prove liveness you are almost doomed to use temporal logic.

pron 60 days ago [-]

I'm not sure what specific temporal properties you're interested in, but safety properties proven through the standard proof techniques (like inductive invariants), and at least common statements about liveness are supported.

> Liveness is about infinite number of states, but any real model checking is bounded by definition.

First, model checking is not bounded to a finite number of states in principle, and certainly not by definition[1]. Even in practice, modern, state-of-the-art model checkers do support infinite state spaces[2] -- of course, not in general, but in some cases. However, TLC, the model checker packaged with the TLA+ tools, is indeed not such a model checker, and can only check finite state spaces; it is what's known as an *explicit state* model checker. Second, liveness is not about an infinite *number* of states, but infinite behaviors, i.e. infinite *sequences* of states; the two are *not* the same.

> but to prove liveness you are almost doomed to use temporal logic.

Oh, absolutely, and after all, the TL in TLA stand for temporal logic, but the point is that proving complex liveness properties in real-world software systems is rare, simple liveness seems to be supported by TLAPS (although most people just use the model checkers), and complex temporal reasoning hardly ever comes up. That's why supporting complex temporal reasoning -- if it is indeed missing in TLAPS -- is not a priority. The features of TLA+ are in general very much driven by the needs of engineers working on real systems.

However, if you happened to come across a liveness property you wanted to but couldn't prove with TLAPS, I'd love to hear about it.

[1]: The name model checking comes from the model checking problem in logic, namely *checking* that a structure M (a program in the case of software) satisfies a logical proposition φ, i.e., that M $\models \varphi$, or that M is a *model* of φ. The name "model" comes from a logical model, i.e., a structure that satisfies a theory or a proposition.

[2]: E.g., https://cpachecker.sosy-lab.org/

Aclassifier 54 days ago [-]

> Well I respect Mr.Lamport`s point, but to prove liveness you are almost doomed to use temporal logic.

I thought CSPm/FDR4 proved liveness on infinite event trails without temporal logic? Can't one extract "never" and "sooner or later" by just exploring the whole state space? Deadlock/livelock will never happen etc.

unboxed_type 53 days ago [-]

Yes, but what if you are unable to explore the whole state space due to its unboundedness? Then the only way I know is by deducing properties thru temporal reasoning.

tmccrmck 61 days ago [-]

Outside of distributed systems what is it used for?

I've only ever heard of it's use in the formalization of distributed systems.

ahelwer 61 days ago [-]

You are correct the language really shines when there is concurrency (not necessarily just distributed systems), because validating system invariants with the model-checker is just magical. I've also found the language to hold great value for specifications as such; Dr. Lamport seems to believe that most technical problems just disappear when you precisely identify what it is you want to do, and writing a TLA+ spec is an efficient way of accomplishing that. So, I've found it useful to write specs for solutions to very technical problems simply to clarify my thinking and root out areas of sloppiness.

ausimian 60 days ago [-]

I concur with both these statements, but would widen the former slightly to say that it helps you think (and then validate your thinking) about any non-trivial state-machine - it doesn't have to be concurrent or distributed.

[Used to work at Microsoft, used TLA+ successfully during product development]

pron 60 days ago [-]

Indeed, TLA+ is mostly known for formalizing and verifying distributed systems and concurrent algorithms. This has a few reasons:

1. Lamport's algorithmic work is in concurrent and distributed algorithms, and as he uses TLA for his own algorithms -- and he's TLA+'s first user -- that's how it's known.

2. Few other *general* software verification formalisms are able to handle concurrency as easily as TLA, so that is where it shines in comparison. Of course, this is intentional because Lamport designed TLA+ to work well for the kinds of algorithms he's interested in.

3. When engineers write software systems that are too complex or subtle to be obviously correct, and could therefore benefit from formal verification, it is usually the case that a concurrent or distributed algorithm is involved.

Having said that, there is nothing specific about TLA+ that makes it any less suitable for sequential algorithms. If you write one that you think is complicated or subtle enough to require help in reasoning, you could certainly benefit from TLA+.

But if you're interested just in sequential algorithms, you do have more options, like Why3/WhyML, which I think is *very* nice (although I prefer TLA+). Whether WhyML or TLA+ would be better suited to verify your sequential algorithm depends largely on personal aesthetic preferences, as well as some details of requirements. For example, WhyML can generate OCaml code, while TLA+ can't. On the other hand, TLA+ has both SMT solvers and a model checker, which makes it more likely that you can verify the entire algorithm automatically, while WhyML requires manual proof in Isabelle or Coq for properties the automatic solvers can't verify (I hear they're working on a proof language directly in Why; I hope it's as nice as TLA+'s proof language). Also, TLA+ lets you describe your algorithm at various levels of abstraction, and show that the more detailed ones implement the more abstract ones. WhyML is more focused on the abstraction level of actual program code.

hwayne 61 days ago [-]

I've personally found it absolutely fantastic for modeling business logic. It's also really good for any kind of concurrent system, which covers pretty much every webtech startup.

a-saleh 60 days ago [-]

I understand that I wouldn't be able to actually run the TLA+ specs, but is there some way how would I be able to run i.e. some of the traces against a live system?

I do QA work and for quite some time I have been intrigued by property based testing. But we are usually testing lot of interconnected micro-services and when I tried to define test-scenario in i.e. quick-check, the end-result was always too complicated to maintain.

I basically tried to do a similar thing to John Hughes "Mysteries of Dropbox" paper.

It seems that with temporal logic it would have been much easier to specify the expected behavior of the program and then use the generated trace to instrument the live service and check that all of the properties still hold.

ahelwer 60 days ago [-]

This is a really good idea! It's been pitched inside where I work in Azure, but not yet implemented anywhere (to my knowledge). The basic idea is you have a TLA+ spec of your system, then you run the model-checker in "simulation" mode to generate random execution traces. You take these generated execution traces, and, according to some predefined correspondence between the actions in your TLA+ spec and some API call in your code (a node becoming inaccessible or a message being sent, for example), run the execution traces on your real-world system.

I'd love to see you implement this and write about your experiences.

sadgit 60 days ago [-]

I havn't looked at the new video series, but when I was trying to get started with TLA+ using the Eclipse toolbox, i got stuck trying to define a behaviour spec for the model checker. Do you know of any good documentation for this? Thanks

cerebraltangent 61 days ago [-]

Kindly add transcripts to this for the benefit of the hearing impaired people ?

ahelwer 61 days ago [-]

Excellent suggestion!

maerF0x0 61 days ago [-]

What role/job title usually writes the TLA+ specs at an organization?

unboxed_type 60 days ago [-]

I hold a "senior researcher" position at research dep. of one of security-related IT companies. I am developing a TLA model to specify one of companies product core protocols. My previous job title was "senior distributed systems developer"; I was doing a similar thing there.

ahelwer 61 days ago [-]

I've yet to come across a TLA+-specific job title, although would love to be surprised. Usually it's the same people writing the spec, or engineers who produce a TLA+ spec from a provided informal-language spec.

qznc 60 days ago [-]

Why not YouTube? Clickable links are possible there as well.

albertocsm 59 days ago [-]

plz check ->
https://groups.google.com/d/msg/tlaplus/lAeWdCE9MLw/v03t3UEv...

justanotheratom 61 days ago [-]

Dr. Lamport does courses at Microsoft? More details please.

ahelwer 61 days ago [-]

Only for Microsoft employees, unfortunately! If you work at Microsoft, send me an email (alias is same as username here) and I'll get you added to the TLA+ Outlook group for future course announcements and such.

scvalencia 61 days ago [-]

That's fantastic. Would be a great contribution to people interested in formal methods. How's your email?

UK-AL 60 days ago [-]

He worked for Microsoft research, so not surprising.

seahckr 61 days ago [-]

attended a course where @ahewler TA'd. can attest, he is awesome! disclaimer - msft engineer.

ahelwer 61 days ago [-]

Aw, thanks!

algorithmsRcool 61 days ago [-]

Introduction @2:49

"What kind of clown am I claiming that I know what can make you think better? ... This is not the time to be modest. I have done seminal research in the theory of distributed and concurrent systems for which i won the turning award. You can stop the video now and look me up on the web. <long pause>..."

rhizome 61 days ago [-]

Hmm, on the face of it that doesn't answer the question.

noblethrasher 61 days ago [-]

The claim is that what he is about to say is worth your *attention*.

Achievement of the highest professional accolade for the very stuff that he plans to discuss is literally *prima facie* evidence in support of that claim.

macintux 60 days ago [-]

The transcription ("turning award" instead of "Turing award") makes it easier to miss the key point.

setheron 61 days ago [-]

I viewed this original when he posted them on the newsgroup. I thought they were very well done, funny and enjoyable. I still haven't applied TLA+ into something at work however I enjoyed learning it nevertheless.

mooneater 61 days ago [-]

Ok I would love to know more about how the RTOS code was shrunk by 10x using TLA+ (at 8:28)

pron 60 days ago [-]

Well, that book costs $99: http://www.springer.com/us/book/9781441997357

But there's been more recent work done in TLA+ on another realtime OS, where the mechanical proof system, TLAPS, was used to prove some aspects of the kernel correct, and you can read about it for free here: https://members.loria.fr/SMerz/papers/abz2016-pharos.html

Aclassifier 54 days ago [-]

I watched this very interesting and amusing series of videos and learned a lot from them. I was shown practical in-use and the motivation for mathematical syntax. I know Promela and CSPm some so I was curious to see more than a trivial example of some C lines becoming a TLA+ spec. I missed real-life examples of down-to-earth usage. The "Formal Development of a Network-Centric RTOS" book shows not only TLA+ but also why in their situation TLA+ was more suitable then other methodologies. Still, on TLA+ I miss the forest for the trees. I probably need more time with Lamport. I hope he's able to do a follow-up. (Or am I unfair here?)

sdbbp 60 days ago [-]

In my experience, writing a formal specification _once_ in TLA+ has shaped my mindset around architecture, implementation, and verification of distributed systems for the last 19 years. It's easier to provide feedback on most informal architecture specifications. It is easier to implement to a specification so as to have a higher confidence of compliance. It is easier to consider the state space of an architecture (distributed system) when in a testing/verification role.

neves 61 days ago [-]

I'm about to start a new system that will have a state machine. The videos are really good to help to think about the problem. This guy didn't get a Turing Prize for nothing:-)

I won't even try to use TLA+ in my business context. Just will search for a good java library. Maybe these videos will help me to recognize a good one?

BTW, do any of my HN's fellows have a good Java state machine opens source library to recommend?

> whitenoice 61 days ago [-]
>
> You could use akka[1] library to build a state machine, actor model is great for building state machines [2]. I have built one in the past using akka. Library is pretty robust and well maintained.
>
> [1] http://akka.io [2] http://erlang.org/documentation/doc/4.8.2/doc/design_princip... [3] http://doc.akka.io/docs/akka/current/java/fsm.html
>
> > neves 60 days ago [-]
> >
> > It looks nice, but overkill for my application. For now I have no need for distributed systems, probably some concurrency in the near future.

colanderman 60 days ago [-]

While TLA+ itself is great, I suspect for many here that PlusCal (a "veneer" on top of TLA+) is more immediately useful. I didn't fully grasp either language until I understood PlusCal for what it is: a simple procedural language with two nondeterministic constructs, which can be used for describing concurrent state machines. Nothing more, nothing less.

(To understand PlusCal, you need first understand the basics of TLA+, but you don't need to understand the action system 100% in-depth.)

The idea behind PlusCal is to write your algorithm in it, leaving out the "unimportant" bits, and using the nondeterministic operators in place of any value that is not in your algorithm's control. The model checker, TLC, can then run all possible traces of your algorithm to search for conditions under which it may deadlock or violate some assertion you have made.

gizmodo59 61 days ago [-]

Just finished the first video. Very nicely done. The website is nicely done too, not obtrusive and makes the learner to focus on the video. +1

rebootthesystem 61 days ago [-]

In many ways this is what APL was originally about. Ken Iverson used it to describe the operation of early IBM computers while at IBM.

> qznc 60 days ago [-]
>
> "A Formal Description of System/360"

Readable (kind of) here: https://www.yumpu.com/en/document/view/40763566/a-formal-des...

rebootthesystem 60 days ago [-]

Nice find!

baby 60 days ago [-]

> TLA+ (pronounced as tee ell a plus, /ˈtiː ɛl eɪ plʌs/) is a formal specification language developed by Leslie Lamport. It is used to design, model, document, and verify concurrent systems. TLA+ has been described as exhaustively-testable pseudocode[4] and blueprints for software systems.[5]

from wikipedia

nchuhoai 60 days ago [-]

Possibly a side question:

What do people think of the format? I have off and on been thinking about a format where the video is the main highlight, but is supported, like this one, with links and other media to make it a more engaging experience. Are there other use cases that you think this would be conducive for?

shaunxcode 61 days ago [-]

These are great so far! I wish you had posted about it once they were all available though as I was fully prepared to binge watch.

Ambrosia 61 days ago [-]

Relevant: http://sam.js.org

Avshalom 61 days ago [-]

So how amenable is tla+ to automatic translation to an existing programming language?

hwayne 61 days ago [-]

IMO not very. It's easy in TLA+ to write things like "randomly select some bijective function from the power set of N to [0, 1]", which can be really handy for speccing but isn't exactly programming-friendly. I personally think of this as a good thing, actually. TLA+ can be simpler and more expressive than it would be if it had to also be translatable.

So you can't guarantee your implementation matches your spec. All the more reason to write unit tests!

pron 60 days ago [-]

That is not such a big problem. Many operators in TLA+ -- like division -- are defined in terms of CHOOSE, but that doesn't stop TLC from implementing them efficiently. It's very easy to supply efficient implementations for various built in constructs, and tell you that you shouldn't use CHOOSE in your own definitions that you want to be automatically translated. So the data and data operations aren't the big problem; the control state is a bigger problem: It is hard to translate TLA+ constructs to programming language idioms like subroutines, loops and threads.

a-saleh 60 days ago [-]

You might want to investigate the Bloom project. That is the only language I know that uses temporal logic as its base while still viable for writing actual running services.

Didn't use it, but watch a few presentations, i.e: https://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/Bl...

ahelwer 61 days ago [-]

Amenable? Probably fairly amenable. It hasn't been done yet, though. It likely wouldn't be translation so much as proving that the existing code refines a TLA+ spec.

pron 60 days ago [-]

Showing that program code refines a TLA+ spec has been done for C and Java (although scalability is a different matter).

C: http://link.springer.com/chapter/10.1007/978-3-319-17581-2_1...

Java: http://ieeexplore.ieee.org/document/6042069

The C work seems more thorough.

ahelwer 60 days ago [-]

Amazing! I hadn't heard of this, thank you. I'll add it to the Wikipedia article.

devdoomari 60 days ago [-]

the video says it's 'video#1'... so where's the link to video#2, #3, ... ???

*edit: sorry I see the link after the end of video#1...