Velocity

V5 2026

Booz Allen

INSIGHTS FOR INNOVATORS

AI vs AI

REIMAGINING CYBER FOR A FASTER FIGHT

The MATH That Makes Technology Trustworthy

Formal methods and automated reasoning are reshaping how agencies can validate critical software, enforce AI guardrails, and eliminate entire classes of vulnerabilities

By Wyatt Chaffee, featuring Bill Vass and Byron Cook

Federal missions increasingly rely on software and AI systems—for example, algorithms and automated decision-support tools—to make real-time decisions under conditions that are fast, fluid, and adversarial. Yet the methods used to ensure the correctness and accuracy of systems and programs have barely changed. Traditional testing can show that software works in some scenarios—but not that it works in all of them. And as AI-enabled systems expand the space of possible behaviors, the gap between what agencies can test and what they must trust is widening.

> “The idea behind formal methods—going back to the Greeks—is that rather than thinking about the content of the argument, you think about the form of the argument. Now you can say: If the argument has this form, it’s correct, regardless of what ‘P’ and ‘Q’ actually are. Automated reasoning is the automation of the reasoning over that form.”
>
> —Byron Cook, Amazon Web Services (AWS) Vice President, Distinguished Scientist, and Automated Reasoning Group Lead

This is where the mathematically grounded techniques of automated reasoning and formal methods shift the landscape. Instead of sampling behaviors through tests, automated reasoning generates mathematical proofs that ensure a system will always uphold its most critical properties—no matter the input, environment, or adversary. It replaces the question “Did we test enough?” with “Do we have a proof?” It’s the difference between stress-testing a bridge by driving trucks across it and demonstrating through structural calculations that the bridge will hold any load intended, while testing remains essential for properties that resist formalization.

Long reserved for chip manufacturers, aerospace systems, and a handful of high-assurance environments, these techniques are now entering mainstream engineering. Cloud service providers use automated reasoning to verify the identity, access, and network isolation policies of their platforms. Defense and space systems rely on it to eliminate entire classes of safety-critical bugs. And as AI becomes central to federal operations, agencies are beginning to explore formal verification as mission environments increasingly require a backbone of trusted autonomy and policy-aligned behavior.

This article equips IT leaders—particularly those responsible for systems with national security implications—with the basic insights needed to assess automated reasoning and formal methods in greater depth. We conclude with an extended Q&A with Byron Cook, AWS vice president, distinguished scientist, and automated reasoning group lead, whose work has moved automated reasoning from academic theory to mission-critical capability protecting billions of secure transactions each day.

BEYOND TESTING

Conventional software testing works by running a program against specific inputs and checking whether the outputs are correct. Does the login screen accept a valid password? Reject an invalid one? Handle a blank field?

Each test covers a single scenario. However, even a simple program can accept many possible inputs, and testing can cover only a tiny fraction.

Consider a function operating on 32-bit rational numbers. Testing every possible input would require time-consuming computation. By contrast, a formal methods tool can represent the entire set of 32-bit rationals symbolically and prove the desired property for all of them in minutes or seconds. While testing scales with the number of cases checked, formal reasoning scales with the overall logic of the problem.

Working in this way, an automated reasoning tool can examine an authentication module and produce a mathematical proof for which no input—no matter how malformed or adversarial—will ever lead to unauthorized access. Compare this with machine learning. Machine learning identifies statistical patterns and makes probabilistic predictions. Automated reasoning operates on logical rules and produces deterministic, provably correct conclusions. When an automated reasoning tool reports that a property holds, that result is mathematically guaranteed—not for most cases, but for all cases. It is as certain as a geometry theorem.

KEY CAPABILITIES, UNDER THE HOOD

Computer scientist Tony Hoare created a framework for proving program properties in 1969. For roughly the next two decades, the field of formal methods advanced steadily but remained specialized. In 1994, Intel’s Pentium processor was found to have produced incorrect results for certain floating-point division (FDIV) operations. The FDIV bug cost approximately $475 million in recalls. As a result, formal methods increasingly moved toward becoming a valued engineering component. Advances in solver technology soon followed.

While formal methods proofs were once done by hand, modern automated reasoning sits on a technology stack encompassing different tools that streamline specific parts of the process, from satisfiability (SAT) solvers and model checkers to theorem provers.

SAT solvers determine whether any combination of Boolean values can satisfy a logical formula; they handle problems with millions of variables. For example, you can use this to automatically verify that no combination of configuration settings in a complex system will produce a contradictory or unsafe state. **Satisfiability modulo theories (SMT) solvers**—the most widely used is Z3, developed at Microsoft Research—extend this to integers, arrays, bit-vectors, and other data types to enable the higher-level verification tools engineers use. This tool helps you confirm, for example, that a memory allocation routine can never return a negative size value, regardless of what inputs the program gets.

Built on top of these are **model checkers,** which look at every state a system can reach, a technique helpful for systems where timing bugs are difficult to find. You can use this to verify that two processes in a network protocol can never simultaneously believe they hold an exclusive lock. **Interactive proof assistants**—such as Rocq (formerly “Coq”), Isabelle, and Lean—let a human construct a formal proof while the computer verifies every logical step, producing a high-assurance artifact. This tool can help you formally certify that a compiler or cryptographic library behaves exactly as specified across all possible inputs, via a machine-checked proof that humans can inspect and audit at every step.

Two additional techniques extend the toolkit. **Abstract interpretation** enables static analyzers to reason about program properties but not run the program. For example, you can use this to scan avionics software for the entire class of runtime errors before the code executes on hardware. And **symbolic execution** runs programs with symbolic rather than concrete values, exploring all execution paths to identify vulnerabilities. Symbolic execution was demonstrated at scale in combination with other techniques in the Defense Advanced Research Projects Agency (DARPA) 2016 Cyber Grand Challenge.

These tools require **formal specifications**, which are precise mathematical descriptions of what a system is supposed to do. Specification languages include computer scientist Leslie Lamport’s TLA+ (Temporal Logic of Actions), used for distributed systems, and the Frama-C/ACSL framework for verifying C programs in avionics and nuclear safety. However, it’s important to keep in mind that a formal proof is only as trustworthy as the model it verifies. If the specification doesn’t faithfully represent the actual code, the engineer may have proven the wrong thing, leaving the software unverified. This is why automation that enables tracing across specification, proof, and implementation is more reliable than manual transcription, which can introduce the risk that a tacit assumption has broken the chain of evidence.

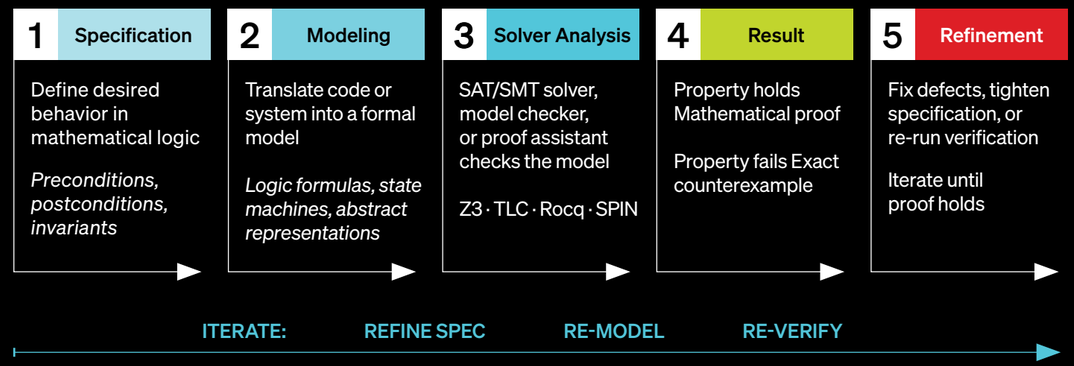
CODE THAT BUILDS TRUST

Wherever a single software error can be catastrophic—such as flight control, cryptography, autonomous vehicle operation, and financial transactions—automated reasoning and formal methods are gaining relevance.

For example, a team in France built the first formally verified C compiler. The mathematical proof underlying the CompCert tool guarantees that compiled code keeps the meaning of the original source. A team in Australia produced the seL4 operating system kernel, which has been formally verified. In the aerospace sector, the Astrée tool confirmed that runtime errors were not present in

From Source Code to Proof:

The Formal Verification Workflow



Key insight: When the solver finds a violation, it does not just report “fail.” It produces an exact counterexample—the specific input, state, or execution sequence that breaks the property. These counterexamples often reveal bugs that years of testing never found.

When the proof succeeds, the guarantee covers all possible inputs—not a sample, not a probability, but a mathematical certainty.

Airbus A340 and A380 primary flight control software. And the Certora Prover applies formal verification to smart contracts on blockchain platforms, where a logic error could cause hundreds of millions of dollars in losses.

However, not all code should be verified through automated reasoning given the significant tradeoffs in terms of time, cost, energy usage, and skillset requirements when compared with conventional testing. Questions to ask include: Would failure be catastrophic? Would it create regulatory exposure? Is the code path small but highly consequential—a ledger update, an authorization decision, a cryptographic handshake? Most organizations will not verify their entire codebase. They will identify high-stakes components and apply formal methods selectively while relying on testing for the rest.

It's also important to note that, even when an organization identifies a strong use case, it may lack the expertise to carry it out. Formal methods require foundations in mathematical logic, automata theory, and programming language semantics. GenAI promises to democratize these tools to a greater extent in the future. For example, LLM-based chatbots can assist in writing specifications and translating natural-language requirements into formal models.

PROOF OF PERFORMANCE

The federal government has supported formal methods for decades. DARPA's High-Assurance Cyber Military Systems (HACMS) program adopted seL4 as the foundation for formally verified software on unmanned military platforms. Starting with a small quadcopter and then moving to the Little Bird helicopter, the program used formal methods to build software that creates machine-checkable proofs. DARPA has applied the HACMS tools to other military systems, including satellites.

The National Security Agency has required formal specifications under the Trusted Computer System Evaluation Criteria (i.e., the Orange Book). At the highest assurance class, A1 (Verified Design), the Orange Book required a Formal Top-Level Specification of the trusted computing base, configuration management procedures enforced throughout the system lifecycle, and formal verification methods to show that the implementation is consistent with the design. Although the Orange Book was later superseded, it established the concept that

> "What I really loved about automated reasoning is that it's like little mechanical engines of truth."
>
> —Byron Cook, Amazon Web Services (AWS) Vice President, Distinguished Scientist, and Automated Reasoning Group Lead

The Trust Stack: Where Automated Reasoning Sits in AI Deployment

The key principle: The AI model remains creative and capable. The formal layers above it ensure it cannot violate policy. The model proposes; the logic disposes.

LAYER 4
User/Mission System
Receives only verified, policy-compliant outputs. Mathematical certainty for evaluation, not statistical confidence.

LAYER 3
Policy Specification
Formal declarations of allowable and disallowed behaviors. Written in languages like Cedar, TLA+, or Alloy.

LAYER 2
Formal Verification
SMT solver or reasoning engine checks every output against declared rules before it reaches the user.

LAYER 1
AI Model
Generates outputs based on statistical patterns. Non-deterministic: identical inputs can yield different outputs.

AUTOMATED REASONING TRUST LAYER
Provably correct guardrails ensure AI outputs comply with formal policy before reaching the user.

system trust requires mathematical verification. In addition, NASA's Formal Methods Research Group includes participants from six NASA centers and holds an annual symposium on formal techniques for software assurance in space, aviation, and robotics. NASA Ames has applied model checking to verify autonomous planning software, including planning systems flown on Deep Space 1.

AWS has led the commercial deployment of these techniques after assembling a team of automated reasoning researchers and applying formal methods to its cloud infrastructure. A tool called Zelkova uses SMT solving to analyze identity and access management policies, powering IAM Access Analyzer for AWS customers. Another tool, Tiros, verifies network configurations. AWS engineers used TLA+ to find bugs in DynamoDB and other services.

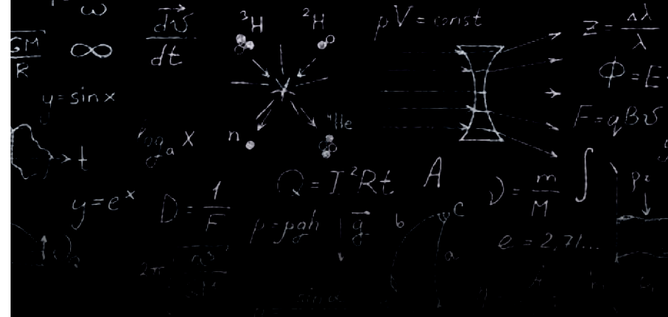
More recently, AWS extended its approach with Cedar, an open-source policy language backed by an automated reasoning engine. Cedar lets developers write access control rules that are easy to understand. The reasoning engine verifies, at deployment or runtime, that policies are internally consistent and that no combination of permissions creates an unintended access path.

A MISSING PIECE FOR TRUSTWORTHY AI

LLMs are non-deterministic by design: even identical inputs can yield different outputs. In addition, they sometimes hallucinate. For intelligence analysis, medical decision support, financial compliance, and other critical missions, automated reasoning offers a way to manage these attributes. Although automated reasoning can't explain how a model thinks, it can enable engineers to constrain what it is allowed to do. They can use formal methods to declare allowed and disallowed behaviors, then enforce those limits. As a result, the model can function productively without violating policy.

In **LLM-guided formal verification**, a language model generates a candidate proof, and a formal tool checks it. If the model hallucinates, the verifier catches it. Google DeepMind's AlphaProof system used this approach in 2024 to solve International Mathematical Olympiad problems at a level competitive with silver-medal finishers.

In **formal guardrail architectures**, automated reasoning acts as a runtime verification layer; it checks every LLM output against formal rules before it reaches the user. Does this query violate access control? Does this dosage exceed safe limits? In **verified code generation**, formal tools verify AI-generated code before deployment, which is critical when human-in-the-loop review can't scale to the volume of code AI produces.



A Brief History of Automated Reasoning

From Mathematical Logic to Industrial Deployment

Foundations (1956–1985) →

1956–1969: The Birth of Automated Reasoning

Newell and Simon's Logic Theorist (1956) demonstrated automated proof search. Building on earlier work by Davis and Putnam, the DPLL algorithm (1962) laid the foundations of modern satisfiability solving. Hoare logic (1969) introduced a mathematical framework for reasoning rigorously about program correctness.

Amir Pnueli's introduction of temporal logic (1977)

Enabled the precise specification of concurrent and reactive systems. In the early 1980s, model checking was developed by Edmund Clarke and E. Allen Emerson, and independently by Joseph Sifakis, introducing automated, exhaustive verification for finite-state systems.

Proof Assistants and the SAT Revolution (1979–2010) →

1979–Early 2000s: Theorem Provers Come of Age

NQTHM (1979) marked major advances in practical automated theorem proving. By the late 1980s, proof assistants including Isabelle and Coq enabled expressive higher-order reasoning with machine-checked guarantees, maturing into powerful environments for large, mechanically verified proofs.

1995–2010: SAT and SMT Transform Verification

Satisfiability solving underwent a breakthrough in the late 1990s with conflict-driven clause learning (CDCL), making SAT—and later SMT—solvers practical engines for software and hardware verification. The SLAM project led to the Static Driver Verifier (introduced 2004, shipped 2006), one of the first large-scale industrial deployments of automated model checking, applied across the Windows driver ecosystem. Z3 (2007) unified logical reasoning with theories of arithmetic, arrays, and data structures, becoming a foundational tool across academia and industry.

Industrial Deployment (2005–2017)

Mid-2000s: CompCert and Verified Compilation

CompCert (2006), developed by Xavier Leroy, was the first widely recognized, practically usable verified compiler—carrying a machine-checked, end-to-end proof of semantic preservation and showing that formal methods could produce deployable systems.

2009: seL4 and Operating System Verification

The seL4 project achieved the first full formal proof of functional correctness for a general-purpose OS kernel, developed using Isabelle/HOL and subsequently deployed in security- and safety-critical domains.

2015–2017: Cloud-Scale Formal Verification

s2n (2015) was formally analyzed to ensure security properties, with similar techniques applied to distributed infrastructure serving millions of users—demonstrating that formal verification could operate at cloud scale.

Expanding Frontiers (2016–Present) →

2016–2021: Formal Methods in Blockchain and Mathematics

KEVM (2017) provided a formal semantic foundation for reasoning about smart contracts. The Lean ecosystem—including mathlib (2017) and Lean 4 (2021)—enabled large-scale collaborative formalization of mathematics.

2023: Authorization System

Cedar (2023) demonstrated that formally grounded reasoning about access control policies could operate efficiently in real-time production environments.

2024 and Beyond

AlphaProof (2024) generated formally verified proofs for several IMO-level problems, pointing toward AI systems that produce machine-checkable correctness guarantees alongside their solutions.

A PATH TO ADOPTION

Start with automated reasoning, not full formal verification. Full verification—interactive proof construction with tools like Rocq or Isabelle—requires deep mathematical expertise and significant upfront investment. Automated reasoning encompassing solver-backed analysis with Z3, TLA+, Alloy, or Cedar is more accessible, more automatable, and already integrated into commercial services that abstract away the underlying complexity.

Identify highest-stakes components and pilot on a small, high-impact use case. A cryptographic code path, an authorization module, or a state machine controller is where formal methods can deliver the most value relative to effort while multiplying early wins.

Integrate solver-based checks into CI/CD pipelines and invest in developer education. Automated reasoning maximizes return when it runs continuously alongside existing tests, rather than as a one-time exercise. Sustaining that integration requires building developer fluency in foundational CS theory and solver-based workflows.

Build internal exemplars that demystify the discipline. End-to-end, realistic examples are what convert skeptics and give engineering teams a concrete model to follow.

Move now to harness mature capability. Automated reasoning is now entering broader adoption. AI-augmented reasoning—where LLMs generate specifications, translate requirements into formal models, and produce candidate proofs—is also emerging fast. Organizations that build capability now will be positioned to leverage the candidate proofs as guardrails for AI-generated content while exploring additional impactful mission use cases sooner.

LOOKING AHEAD

Language models will continue lowering the expertise barrier for formal methods. Automated reasoning will be embedded in development pipelines, flagging bugs and serving as the trust layer for AI-generated code, recommendations, and decisions. Federal procurement is already reflecting the shift, with formal methods appearing in some acquisition requirements for defense, aerospace, and advanced cloud systems.

Across these developments, a basic concept will continue to be powerful: automated reasoning moves software assurance from “we tested it, and it worked” to “we proved it correct, for all possible conditions, mathematically.” For anyone responsible for systems critical to missions that can’t fail, this difference will increasingly matter in demanding real-world environments.

Reasoning About the Infinite:

An Interview With Byron Cook

To explore where the field is heading—and how automated reasoning is being applied at scale inside major cloud and AI platforms—Booz Allen Chief Technology Officer Bill Vass spoke with Byron Cook, AWS vice president, distinguished scientist, and automated reasoning group lead, to explore new perspectives on automated reasoning and formal methods.

Q Bill: What started your interest in automated reasoning and formal methods?

A Byron: I studied to be a VW [Volkswagen] mechanic in high school – I restored air-cooled 1960s-era VWs. I worked with a retired NASA mechanical engineer at a high school where you didn't get grades and you didn't go to class. You designed your own curriculum, and one of the areas I decided to work on was learning how to rebuild cars.

Then I went to Evergreen State College in Olympia, Washington. There was the world expert on Hilbert's epsilon operator, who taught only advanced logic. He taught a course called "Computer Building Cognition," a year-long course where you just worked with him. These days we'd call it "neurosymbolic AI." I basically went to the registrar at Evergreen and said to sign me up for the hardest thing available, because I just wanted whatever the hardest thing was. They sent me to that course, and I absolutely loved it. This was around 1993 or 1994. Then they said, well, maybe you should do a Ph.D. And I've just never looked back.

What I really loved about the VWs was how simple the motors were. Because of the constraints they were designed under, they minimized the parts you needed. And what I really loved about automated reasoning is that it's like little mechanical engines of truth. You're reasoning about the infinite but using a composition of very simple parts that are very, very powerful. I see a lot of parallels between real mechanical motors and mechanical reasoning about truth.

Q Bill: What are formal methods, what is automated reasoning, and how do they relate?

A Byron: They're cousins. "Formal methods" is the term people used in the seventies and eighties before the tools were very automated. Automated reasoning is the automation of the reasoning you were doing when you did formal methods manually.

The idea behind formal methods—going back to the Greeks—is that rather than thinking about the content of the argument, you think about the form of the argument. Now you can say: If the argument has this form, it's correct, regardless of what "P" and "Q" actually are. Automated reasoning is the automation of the reasoning over that form.

The challenge is that the problem is undecidable. We know that from Gödel, and Turing directly connected it to the halting problem. So, you have to acknowledge that the tools will sometimes just go to lunch: they can't always give you an answer. You have to expect that. Formal methods were the very early days, when the tools weren't there, so you were essentially saying to a human: we don't know how to solve this problem, so we'll let you use human ingenuity, but here's the shape of the solution. You were exploring that space with very little tool support.

Q Bill: How would you explain automated reasoning to a non-mathematician and a non-programmer?

A Byron: Usually, people have done just a little bit of algebra. They've pushed symbols around—X plus Y is the same as Y plus X. You've moved symbols, so X and Y are symbols; that's symbolic reasoning. And I think everyone can look at X plus Y and Y plus X and say, yeah, it's roughly the same thing. But you've actually just reasoned about the infinite—X and Y range over potentially all the integers, all the rationals.

There's some equipment you can pull in to ensure that X plus Y equals Y plus X, and they're the very same mechanisms used around 300 BC to show that the Pythagorean theorem held. Reasoning, in the sense I'm talking about, is finding an air-tight argument as to why something is true, where each step of the argument rests on axioms, or the foundations we've agreed on as a society for how to establish truth. The automated part is just algorithms over that. You're searching for finite arguments about the infinite.

Q Bill: When did you realize you could take this from academic work to actual implementations in production? When did you think it would be practically applied to important parts of code?

A Byron: I was very fortunate that from the very beginning I was right in the middle of application. There was the Intel floating point division bug in the early 1990s, and the Intel Pentium team hired formal reasoning people to prove the correctness of the underlying reorder buffers, register alias tables, and the microcode that implemented floating point arithmetic. I was an intern there as a Ph.D. student.

Then for various reasons I got pulled in when Microsoft realized around 1999 to 2000 that they were at risk of losing out in the data center because Windows was crashing so much—predominantly because of device drivers. I reported to the person who owned the I/O [input/output] subsystem of the kernel and applied these tools there.

After 10 years in pure ivory tower research, when I joined Amazon, I was actually coming back to a familiar space. I repeated what I had seen work at Intel: I'll start four things, message it as though any one of these succeeding is a win, and we'll see where it goes. That complemented Amazon's view of "do something small, see if it works, then double down relentlessly." That approach has just grown like a weed using relatively standard Amazon mechanisms, seeded by how I'd done things at Intel.

These days a lot of what I spend time on is talking to other Amazon leaders because they all want to start science in their own organizations. I talk them through how I framed it, how I measured it, and how to bring science into a pretty delivery-oriented organization.

Q Bill: Amazon has been a pioneer under your leadership in applying automated reasoning in areas like cloud security and policy compliance. What are your key lessons learned from the journey? For example, how have you addressed the costs involved with implementing the technology?

A Byron: Automated reasoning is largely a CPU activity, not a GPU activity, so it's actually relatively cheap. But culturally it's expensive, because the familiarity and intuition about where these techniques will and won't work are not commonly held in the IT community.

You have to think about things differently. Automated reasoning is like jazz. Miles Davis said jazz is all about the notes you don't play. It's all about how you reframe a question into a different question that you can answer efficiently, even though you're reasoning about the infinite without actually running the program or doing all the computation.

In terms of where I focused at Amazon early on: I mined the internal database of all security incidents for situations where my tools would help, prioritized them, matched the problems to the people I knew I could hire, and found four or five promising things to start. From there I was essentially an ambulance chaser at the Friday security meetings with Andy, where they'd bring in the top security close calls of the week and discuss what mechanisms to put in place. I'd help people navigate how they could be using my tools as they went in or came out of those meetings. That built trust, and from there it was a flywheel.

We now have around 350 Ph.D.s working across 20 teams at Amazon, with roughly 150 Ph.D. student interns last year. Automated reasoning has the most value where the algorithms are really hard to get right, or where the consequences of getting them wrong are absolutely catastrophic. That tends to be the lowest levels of the stack and the tight loops—cryptography, the virtualization layer, durability for storage.

And then with agentic AI, what's happening is we're trying to replace a lot of our sociotechnical mechanisms with automation. But agents aren't sentient. They're not afraid to get fired or go to jail, so the mechanisms we relied on before don't hold anymore. You either require a human to approve everything, or you have to put other mechanisms in place to let agents roam. That's becoming an important area for automated reasoning to scale.

Q Bill: How do you envision AI impacting automated reasoning?

A Byron: There's a really exciting chocolate-and-peanut-butter moment happening with agentic AI. A lot of the heuristics—the tools—in automated reasoning were either fully automated but with limited capabilities, or much more powerful but requiring a Ph.D. to guide them. That was a neurosymbolic system where the "neuro" was someone's brain, because they saw the patterns and narrowed down the infinite branching directions of a proof search to the most likely successful strategy. Now we're able to bring in language models to help guide the search for the proof. So, we get 100%- assurance of what's true and untrue, but we also get much more automation. That's opened up some pretty remarkable opportunities.

Q Bill: If you're a CIO or CTO in the government or a big corporation—not building virtual machines or security code or an identity management system—where would you recommend applying automated reasoning?

A Byron: A lot of people right now are looking at the agentic space, and there are a bunch of startups, and Amazon is working here, too. The idea is defining the envelope of behaviors you want to allow and don't want to allow from your agents. Those behaviors define availability, confidentiality, integrity, sovereignty, privacy—the different dimensions and the lines you really don't want agents to cross. You create an envelope, and as long as the agents' behaviors stay within it, you're good. If they try to exceed it, you stop them. That's a fast-moving area. From an Amazon product perspective, that's where something called Agent Core Policy is headed right now.

But if you're not yet doing agentic systems and you're a CIO, I would look at what are the algorithms where, if they're wrong, you're under existential threat. Organizations are often based on trust, and there's some dimension of trust that they're differentiating on—the reason customers come to them. So, you want to get those algorithms right. If it's cryptocurrency contracts, you want customers to be able to trust that the algorithms executing those contracts on their behalf don't have bugs.

On the compliance side, it's very tricky because there's a compliance industrial complex that's ripe for disruption, but it won't be easy. It's less about formally proving everything and more about providing transparency about the decisions you're making and how you know your systems are correct and building trust on that. The mechanisms by which we decide compliance will hopefully adapt to the tools.

It'll be very hard to formally prove that your assessment was correct. But it'll be much easier to show that the controls around tokenization are correct, or that differential privacy is correct, or that the virtualization is configured correctly such that you have an argument as to why something is true—and that argument could be audited every hour instead of every year.

Q Bill: Could you talk about the goals of the work you're doing with DARPA?

A Byron: I had never worked in government but thought I should do a bit of a tour of duty to contribute back to the country that's created such a great environment to live and do work in. I'm trying to help the government figure out what some of the big scientific levers are.

I'm framing this as imagining a world where proof is just a thing. It's pervasive. All the major players, the hyperscalers, and a whole load of startups are bringing automated reasoning and neurosymbolic tools to the table. Lean theorem prover and a lot of these tools are open source. So, imagine all that capability exists. I'm leading exercises to really accelerate the nation's ability to leverage the existence of that capability. The programs I'm brewing up mostly have that flavor: Proof is a thing, now what?

Q Bill: If we had this conversation again in three years, how would you want it to look? What would be your target around automated reasoning, agentic systems, and policy?

A Byron: There are a bunch of things happening right now that I'm really excited about. As a society, we're going through an "art of the possible" exercise. Language models came out of nowhere from the perspective of the general public, and suddenly people had access to easy-to-use tools that could answer a different set of questions and solve a different kind of problem. That has brought neurosymbolic AI along with it. If you look at the Gartner Hype Cycle, neurosymbolic AI—automated reasoning combined with language models—is shooting up in interest.

One of the traditional barriers to entry in this space was step one: you need to write down in logic what you want. That was a non-starter for 99% of the population. But there's interesting work on auto-formalization—building a logical twin of your natural language statement. Imagine you say, "I want all data at rest to be encrypted" or "I never want to log credentials." You can now encode that in an appropriate formalism like linear temporal logic and then use the logical structure to help people who don't understand logic see all the weird corner cases—to get comfortable that the formalism is actually saying what they wanted to say. Then you can maintain it.

I think we're getting very close to allowing a much larger set of people to write down what they want for their systems and argue about it meaningfully in meetings: "No, this is what availability should mean." "This is what durability should mean."

The other piece that's always been tricky with automated reasoning is writing down and getting confidence around the assumptions. When you prove a program correct, you're assuming the runtime is correct, the compiler is correct, the microprocessor is correct, the network is correct. But now you can automatically compute those abstractions and use model-based testing to harden them and then prove them if you see fit.

The interplay between testing, proof, and natural language logic means a lot of the barriers that were so fierce before are melting away. People now have a whole plethora of tools and different levels of assurance they can seek, with different time and energy investments. It provides a menu. It's a pay-as-you-go model, as opposed to becoming a true believer in one technique. The lines between these different techniques, which people used to argue about, are blurring, and people are partnering together more. I'd love to see that continue over the next three years.

We're getting close to the point where you can just say: On this system, I want to make sure these critical things are always true—no privilege escalation, no two planes on the runway in the same place at the same time, never a negative balance—and go prove to me that the code enforces them. And you can generate something that convinces both a theorem prover and an auditor. That translation between formalisms and different people's backgrounds—from formal methods to auditor—is now in reach.

Bill Vass is Booz Allen's chief technology officer, and Wyatt Chaffee is a director of technology innovation. Byron Cook is a vice president and Distinguished Scientist at AWS, where Byron leads the Automated Reasoning Group.

SPEED READ

Automated reasoning uses mathematical proofs to validate that software and AI systems will function as intended, helping eliminate entire classes of vulnerabilities before deployment.

These techniques—already securing cloud infrastructure, aerospace systems, and cryptographic code paths—are emerging as a solution for AI assurance, enabling engineers to define and enforce the boundaries of safe and policy-aligned model behavior.

In our concluding interview, Amazon's Byron Cook describes how automated reasoning has scaled from academic theory to protecting billions of cloud transactions daily, and why neurosymbolic AI is accelerating the field toward broader, mission-level adoption.

Booz Allen®