Treasury Proposal: Polkadot Virtual Machine (PVM) Formal Specification Development

Proponent: 16MHmxX3ZFB1oWboA4XxMTowkSnKH3dKuesQx7oEv7otsLBo

Date: 13.08.2024

Requested DOT: 371,875 USD | 371,875 USDC

Short description: Develop a formal semantics of PVM (Polkadot Virtual Machine) using the K Framework, ensuring robust verification and conformance testing for Polkadot developers.

Project Category/Type: Network Security

Previous treasury proposals: 2024-04-23-Polkadot-Treasury-KMIR

1. Context of the proposal

JAM and PVM

The Join-Accumulate Machine (JAM), described in the Gray Paper, integrates the robust, scalable architecture of Polkadot with Ethereum-like smart contract functionality, creating a hybrid, decentralized, and trustless environment.

PolkaVM (PVM) is a next-generation virtual machine for JAM based on a variant of the RISC-V RV32EM ISA with some minor modifications to better suit the JAM chain. A JIT compiler translates PVM directly to x86 for super fast execution, achieving 100x compilation time speedup over the existing WebAssembly-based approaches with equal or better execution performance.

Formal Semantics

Currently, there exists one prototype reference implementation of the PVM toolchain, but no formal specification. We believe that producing such a formal specification is crucial for ensuring the correctness, reliability, and security of PVM while also encouraging a diversity of client implementation, with particular value for the JAM bounty program. Additionally, it will lay the foundation for future developer tools, cross-chain interoperability, and enhanced governance within the Polkadot network. This foundational work supports Polkadot's long-term goal of becoming a leading multi-chain ecosystem, enabling a wide range of decentralized applications and services.

K Framework

The K Framework offers an environment for defining formal semantics of programming languages or VMs in a user-friendly, modular, and mathematically rigorous manner. From a single formal definition, K is able to automatically generate a suite of correct by construction language tools, including interpreters and formal verifiers. Because of its language-generic approach, any language formally defined in K can immediately take advantage of the years of engineering effort put into making these language tools fast, correct, and approachable. A K definition can then quickly enhance the reliability and security of software systems, making it particularly valuable for blockchain infrastructure and smart contracts.

Proposal Overview

This proposal outlines a comprehensive approach to developing KPVM, a formal semantics of PolkaVM using the K Framework. The engagement is designed to span approximately 4 months with 2 full time engineers, with possible future work to build PVM tooling on top of the formal semantics proposed here.

Explicitly, by formalizing PVM in K, we gain several significant advantages which will increase security and community trust in the Polkadot ecosystem, with immediate value for the JAM bounty program:

- **Client Diversity:** Formal semantics provide a single source of truth for PVM's expected behavior not tied to any particular implementation. This eases the development of new PVM clients, e.g., in the context of the JAM bounty, by allowing developers to refer to a precise but high-level semantics rather than deal with the complexities of a particular reference implementation.
- **Conformance Testing**: With a formal model, and the corresponding reference interpreter generated from this model, we can conduct thorough conformance testing of different PVM implementations, such as the PVM-to-x86 toolchain or again submissions for the JAM bounty, ensuring consistency and reliability across various platforms.
- **Developer Tooling:** Formal semantics provide a solid foundation for developing various developer tools, with the K approach automating much of this process. These tools can significantly enhance the development workflow, making it easier for developers to write, test, and verify their code.

We at Runtime Verification (RV) are particularly well-equipped to complete this task, having already successfully applied these techniques to the EVM ecosystem by creating KEVM, a formal semantics of EVM. Our flagship product, the K Framework, has been

used to successfully analyze and verify software written in a variety of languages including Solidity/EVM, Wasm, Michelson, and more, and our existing semantics for RISC-V RV32EM will enable the rapid development of KPVM. In the past, we have used similar formal semantics to develop tooling including

- **Debuggers**: Tools that allow developers to step through code execution to identify and fix bugs.
- **Property Testing Tools**: Automated tools that generate test cases to verify that the software adheres to its specifications.
- **Symbolic Execution Frameworks**: Tools that analyze the behavior of the software across all possible inputs to ensure correctness and security.

We envision future proposals to develop such tooling based on KPVM.

Additionally, we are in close contact with Parity and the developers of PVM, and after our discussions, Parity encouraged RV to apply to the Polkadot treasury to proceed with this formal specification development. We have expertise in both formal verification *and* blockchain systems, and RV has completed dozens of blockchain audits both inside and outside of Polkadot *and* verified several blockchain consensus protocols.

2. Problem statement

With an increasingly digitized world, software correctness has never been of greater importance. For blockchain applications in particular, with their massively distributed, permissionless nature and intermingling of financial systems and software, absolute correctness is *crucial*. As has been repeatedly demonstrated by numerous exploits, traditional testing techniques and ad-hoc implementations are not sufficient to ensure the security of these ever-growing and subtly complex software systems.

Instead, we must take a more principled approach - precisely and formally defining the intended behavior of our programs, and developing approachable and user-friendly tools and techniques to ensure our programs actually conform to this intended behavior.

Currently, PVM lacks such a formal specification, posing risks in terms of undetected bugs and vulnerabilities, and hindering the development of new PVM implementations. The K Framework provides a perfect environment to address this - allowing precise and executable formal definitions, and providing a well-tested approach to developing tooling on top of this definition. We believe creating a formal semantics of PVM using the K Framework will play an essential role in achieving the required level of correctness, security, and reliability for software systems on the Polkadot ecosystem.

3. Proposal objective(s) or solution(s)

Executive Summary

The objective of this proposal is to develop a formal semantics of PVM using the K Framework. By creating such a formal model, we aim to validate PVM's operations, provide a single source of truth for PVM's behavior as a reference for client implementers, and pave the road for a future suite of development tools, such as debuggers and property testing frameworks and formal verification tools, enhancing the reliability and efficiency of the Polkadot ecosystem.

Concretely, the following are the objectives of the proposal:

1. **Establish a comprehensive PVM test suite**: Establish a suite of tests capturing the expected behavior of the PVM. These tests will help ensure the accuracy of the developed formal semantics, the existing PVM-to-x86 toolchain, as well as submissions to the JAM bounty program.
2. **Develop a formal semantics of PVM in K**: Create a precise, formal model of PVM using the K Framework, helping to ensure PVMs security and reliability, acting as a reference for PVM implementers, and laying a principled foundation for future tooling development.
3. **Provide a human-readable version of the semantics for PVM documentation**: Provide an accessible, in-browser formal specification of PVM based on the K semantics, bolstering the PVM documentation analogous to the KEVM Jello Paper.
4. **Fuzz the PVM-to-x86 Toolchain against the KPVM reference, analyzing both correctness and performance**: Ensure that the PVM-to-x86 toolchain conforms to the KPVM semantics by comparing on randomly generating programs and resolving any discrepancies. Additionally, analyze any unexpected execution performance for particular inputs, e.g., JIT bombs, further increasing confidence in both implementations.
5. **Do a time-boxed development of a full-client in Rust using KPVM for execution**: Develop an initial full-client implementation on top of the KPVM reference to provide an additional reference for the JAM chain.

Follow-up Questions

Q: Why is the tool named KPVM?

A: The name comes from two sources: the K Framework and PVM (Polkadot Virtual Machine). KPVM represents the formal semantics of PVM expressed in the K framework.

Q: How does KPVM integrate with PVM?

A: KPVM involves creating a formal, executable model of PVM using the K framework. This model helps in understanding and verifying the behavior of PVM, ensuring that it adheres to its specifications.

Q: Why focus on formal semantics for PVM?

A: Formal semantics provide a mathematically rigorous way to define and verify the behavior of PVM, ensuring correctness, security, and reliability. This is crucial for building trust in the VM and the applications running on it.

Q: What makes the K Framework uniquely suitable for this task?

A: The K Framework allows for the precise definition of programming languages and VMs. Unlike other approaches, the formal K specification can be executed directly, and automatically generates interpreters, compilers, and verifiers from a single source truth. This ensures a small trust base, improving correctness and as well as the ease of future extension or modification.

Q: How will this project benefit the Polkadot ecosystem?

A: By developing KPVM, we enhance the security and reliability of PVM, providing tools for conformance testing, formal verification, and debugging. This strengthens the overall infrastructure and trust in Polkadot-based applications.

runtime verification

Polkadot

Project Team Members

Engineering Team:

1. **Scott Guest**, **Formal Verification Engineer, FTE**
   Scott Guest is a compiler engineer with a broad interest in programming languages and formal verification. He is passionate about both theory and practical implementation, with experience that ranges from formalizing pattern matching semantics for the live functional programming language Hazel to implementing compiler optimizations in a widely-used MATLAB to C++ code generator. Scott received his B.S. in Mathematics and B.S. in Computer Science from the University of Michigan. Outside of work, he enjoys cooking plants, flying in wind tunnels, and jumping out of planes.

2. **Georgy Lukyanov**, **Formal Verification Engineer, FTE**
   Georgy Lukyanov has extensive expertise in the design and implementation of formal verification tools, and applying these tools to real systems at scale. Georgy holds a PhD in Computer Engineering from Newcastle University, UK, where he worked on a symbolic execution engine for a custom instruction set architecture for space satellites. At Runtime Verification, Georgy works on the K Framework, both as one of the developers of K's symbolic execution backend and as a semantics engineer. In his free time, Georgy enjoys playing guitar, spending time outdoors, and reading science fiction.

Project Management & Advisory team:

1. **Yale Vinson**, **Project Manager**
   Yale Vinson has spent over fifteen years working in the FinTech industry building digital financial services products. Ranging from SIM based payments to tokenized payments, digital gift cards, and peer to peer transactions, he has focused on delivering secure digital financial products that put the needs of consumers first. Most recently, Yale worked on products which enabled crypto for the masses at Diebold Nixdorf and MoneyGram. Yale has a B.S. and an M.S. in Electrical Engineering from Texas A&M University.

2. **Everett Hildenbrandt**, **Technical Advisor**
   Everett Hildenbrandt is a formal modeling engineer and CTO at RV. His interests include automated system analysis via symbolic model checking, rigorous software development via carefully designed development practices, and applying these techniques to the software used in the other sciences (eg. physics, biology). He strongly believes that programming languages and system description languages should not be put together in

an ad-hoc manner, rather they should be carefully designed using state of the art language-building tools.

3. **Gregory Makodzeba, Business Development Coordinator**
   Gregory brings a unique combination of technical education and business-related blockchain expertise. With prior education in Aviation Management from Georgian College and a Bachelors in Computer Science from the National Technical University of Ukraine ‘Kyiv Polytechnic Institute’, Gregory has built a robust foundation in both analytical and engineering disciplines. His entrepreneurial spirit is evidenced by his co-founding role at Rektoff, a community-driven cybersecurity R&D company specializing in Web3 security solutions, where he excelled in forging strategic partnerships and spearheading innovative security strategies. Beyond his business acumen, Gregory has actively contributed to the blockchain community as a mentor at ETHGlobal Waterloo and ETHToronto, guiding participants through the complexities of web3 development and security. At Runtime Verification, Gregory is leveraging his comprehensive background to enhance business strategy, and client engagement and drive the adoption of secure, reliable decentralized applications.

Milestones

Here we provide high-level proposal milestones; see appendix for full details.

Technical Milestones

1. **Map Implementation to Specification (2 FTEs, 3 weeks)**: Investigate the existing PVM implementation to record the intended semantics of PVM, producing a high-level plan for the architecture of KPVM.
2. **Collect PVM Test Cases and Setup CI (2 FTE, 2 weeks)**: Establish a comprehensive suite of test cases for PVM and set up continuous integration (CI) to automate testing.
3. **Develop KPVM Syntax and Semantics (2 FTE, 4 weeks)**: Create a formal model of PVM using the K Framework, ensuring a precise and executable representation of its operations.
4. **Pass All Tests in the Selected Test Cases (2 FTE, 2 weeks)**: Ensure the developed semantics correctly handle all identified test cases, validating PVM's reliability.
5. **Develop In-Browser Formal Specification for Integration into Official Documentation (2 FTE, 1 week)**: Provide an accessible formal specification of PVM integrated within its official documentation.
6. **Fuzz Testing Between PVM-to-x86 Toolchain and KPVM (1 FTE, 5 weeks)**: Verify conformance between KPVM and the PVM-to-x86 toolchain using fuzzing techniques to identify and resolve discrepancies, both in terms of results and unexpected execution performance (JIT bombs).
7. **Develop time-boxed full-client implementation in Rust using KPVM for execution (1 FTE, 5 weeks):** Develop an initial full JAM-client implementation which uses the KPVM reference for PVM execution.

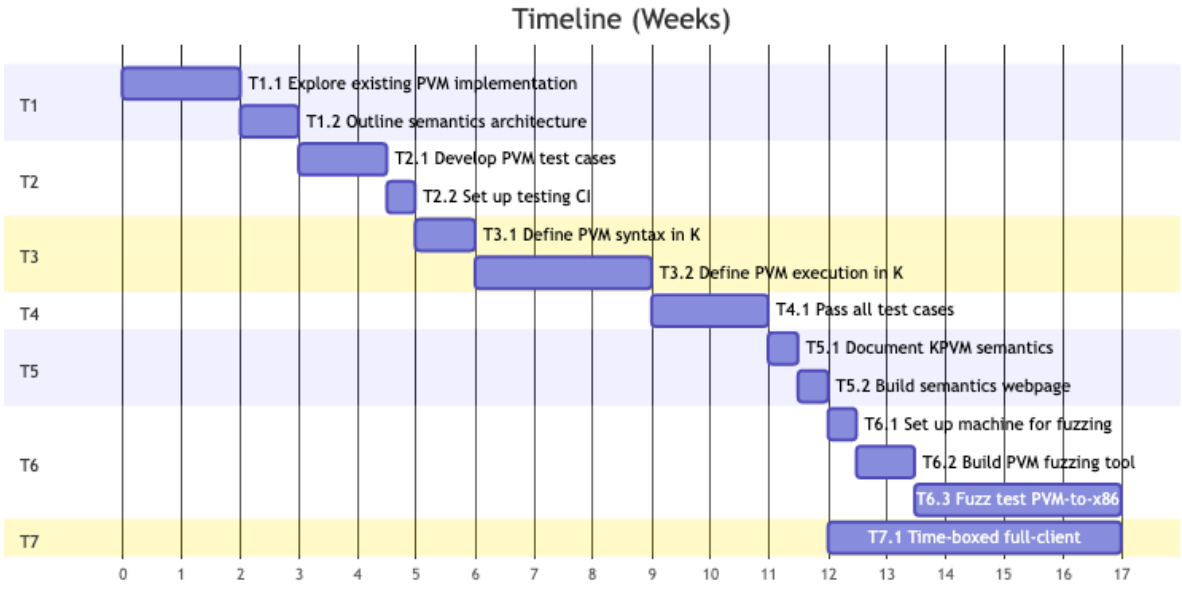
Project Management & Advisory Milestones:

1. **Project Management & Advisory Oversight (1 PM, 1 Technical Advisor, 1 BD Coordinator - 17 weeks - Part-time)**:

   Managing the project timeline, resources, and deliverables to ensure everything stays on track. This includes holding strategic planning sessions, maintaining high standards of quality and ensuring the deliverables meet the expected performance metrics.

Timelines

Our expected timeline is roughly 4 months (17 weeks) with 2 full-time engineers (FTE), a project manager, a technical advisor and some help from an busi/marketing coordinator. Note that, given the large time frame, it's possible that at some point during the project, some engineers will need time off. Where possible, we will assign backup personnel to avoid any impact on the overall schedule.



runtime verification

Polkadot

Budgets

We are billing at the following rates:

- Full-time employees: $25k/person-month
- Part-time employees: $12.5k/person-month

For 2 full-time engineers, 1 part-time project manager, 1 part-time technical advisor and business development coordinator, we are asking for 371,875 USD.

Technical Milestones

1. **Map Implementation to Specification (2 FTEs, 3 weeks):**
   - **Cost:** $37,500
2. **Collect PVM Test Cases and Setup CI (2 FTEs, 2 weeks):**
   - **Cost:** $25,000
3. **Develop KPVM Syntax and Semantics (2 FTEs, 4 weeks):**
   - **Cost:** $50,000
4. **Pass All Tests in the Selected Test Cases (2 FTEs, 2 weeks):**
   - **Cost:** $25,000
5. **Develop In-Browser Formal Specification for Integration into Official Documentation (2 FTEs, 1 week):**
   - **Cost:** $12,500
6. **Fuzz Testing Between PVM-to-x86 Toolchain and KPVM (1 FTE, 5 weeks):**
   - **Cost:** $31,250
7. **Develop time-boxed full-client implementation in Rust using KPVM for execution (1 FTE, 5 weeks):**
   - **Cost:** $31,250

Project Management & Advisory Oversight **(1 PM, 1 Technical Advisor, 1 BD Coordinator - 17 weeks - Part-time):**

- **Cost**: $159,375

Total: $371,875

4. Proposal report

Accountability is an important part of the grant process.

For that reason, each task in our milestones table has an accompanying code or documentation artifact. We will consider the task complete when that artifact is available in our GitHub project repository.

As we complete milestones, we will update the community on our progress via forum posts on Polkassembly or Subsquare with links to any related deliverables. When the project completes, we will prepare a final report documenting our progress using this recommended report template.

5. Payment conditions

This submission represents a unified treasury proposal (i.e., one Treasury proposal that represents all milestones). We are requesting 371,875 USD or 371,875 USDC in total funds to be delivered to Polkadot address: **16MHmxX3ZFB1oWboA4XxMTowkSnKH3dKuesQx7oEv7otsLBo**

The proposer Polkadot account is **16MHmxX3ZFB1oWboA4XxMTowkSnKH3dKuesQx7oEv7otsLBo**.

The project has the following representatives available on Polkadot Direction:

- **Everett Hildenbrandt: @ehildenb:matrix.org**
- **Gregory Makodzeba: @hyperstructured.greg:matrix.org**

6. Comments, Qs&As

At this time, there have not been any public questions or comments.

7. Why Polkadot Network?

Polkadot is a thriving blockchain community, and we are excited to be a part!

8. We'd love to hear about how you got to know about the Polkadot on-chain treasury.

Previously, we were encouraged by Web3 Foundation to apply for a Polkadot Treasury grant for our previous KMIR proposal. For the KPVM work in this proposal, we were encouraged to apply during previous discussions with Parity.

runtime verification

Polkadot

Appendix: Milestones and References

1. Milestones

Technical milestones

Milestones	Tasks	Deliverables	Notes
T1 Map Implementation to Specification	T1.1 Investigate the existing PVM reference implementation to determine the intended semantics	Small document informally describing the semantics of PVM	The deliverable is intended to aid the rest of KPVM development rather than be documentation for public consumption.
	T1.2 Architect a high-level outline of the KPVM semantics	Description of high-level module structure of KPVM semantics	We have an existing RV32EM semantics, so we will also determine whether to fork or directly embed this semantics to model PVM.
T2 Establish PVM Test Suite	T2.1 Collect and develop PVM test cases	A suite of PVM input test files and expected results	
	T2.2 Set up CI	Bare-bones KPVM repository which executes the full test-suite on CI	
T3 Develop KPVM Syntax and Semantics	T3.1 Implement K syntax definitions for human-readable PVM assembly	Working K parser for human-readable PVM assembly	

	T3.2 Define PVM execution semantics in K	K definitions with rules to evaluate each PVM instruction	
T4 Pass All Tests in the Selected Test Cases	T4.1 Pass all tests in the test suite from T2, resolving any failing tests.	CI results that show the full test suite passing	
T5 In-Browser Formal Specification	F5.1 Add explanatory documentation to KPVM semantics	Updated K semantics files as literate Markdown with interspersed documentation	
	F5.2 Integrate documented semantics into user-friendly webpage	Web page for KPVM analogous to the KEVM Jello Paper	
T6 Fuzz Testing PVM-to-x86 Toolchain and KPVM	T6.1 Allocate machine with containerized environment to execute both PVM-to-x86 and KPVM	Container able to execute and compare test suite results between PVM-to-x86 and KPVM	
	T6.2 Build fuzzing tool to generate random PVM programs	Tool which generates random PVM programs	
	T6.3 Fuzz test the conformance of the PVM-to-x86 implementation	Document analyzing any discrepancies discovered through fuzzing, as well as any unexpected execution	One machine to run continuous fuzzing for one month. Done in parallel with the work on T7.

T7 Time-boxed full-client implementation	T7.1 Develop an initial full-client implementation of JAM using KPVM for execution	Time-boxed implementation	The implementation here is time-boxed, and done in parallel with the work on T6 with separate engineers assigned to each task.

Project Management & Advisory Milestones

Milestones	Tasks	Deliverables	Notes
P1	P1.1 Establish and maintain project governance frameworks	Comprehensive project management plan	
	P1.2 Conduct regular advisory and strategic planning sessions	Minutes and action items from meetings	
	P1.3 Quality assurance and control procedures implementation	QA reports and improvement logs	
A1	A1.1 Conduct technical reviews of project deliverables	Detailed technical review reports	
	A1.2 Provide expertise on advanced technical issues	Recommendations report for technology and architecture	
	A1.3 Assist in technology selection and architectural decisions	Training materials and session recordings	