Quint Specifications for Design and Testing

Ivan Gavran[1]

[1] Informal System
https://www.informal.systems

Abstract. We report on specifications developed in Quint, a novel specification language based on TLA. The specifications are developed as a part of Informal System's work on development and security audits of various blockchain systems. In this report, we focus on two specifications: 1) the specification of Malachite implementation of Tendermint consensus, 2) the specification purpose-built for testing of liquidity migration for Neutron's decentralized exchange. We describe the properties of those specifications and provide links to those (and some more) Quint specifications.

1 Introduction

Informal Systems is a tech company working on design, verification, and implementation of distributed protocols. Our main focus has been on blockchain-related tehcnologies: consensus algorithms, light-clients, bridges, and decentralized exchanges.

Over the course of last five years, as part of our work, we have specified a number of systems at different phases of development: in the design phase, testing phase, or auditing phase. Our specifications have been written in TLA+ [1] and Quint [2]. (A full list of systems specified in Quint, by Informal and others, is available on Quint's homepage [3].)

In this report, we want to showcase two example specifications.

The first one specifies Malachite [4], a Byzantine-fault tolerant consensus engine in Rust. We chose the Malachite specification [5] as an example of a specification that was done prior to and alongside the implementation, to help with design decisions and spot problems early. Furthermore, it is a specification of a large, production-ready codebase.

The second specification [6] was written for a decentralized exchange (DEX) running on the Neutron blockchain [7]. It was developed in the context of reviewing Neutron's liquidity migration to new liquidity pools. This was a high-risk operation that, if not done correctly, would jeopardize funds of many users. Thus, we created a simple specification that was then used to create end-to-end test to run against. Using that specification, we managed to find an interesting bug, resulting in one user getting funds of the others.

In the workshop, we plan to give only a brief overview of the first specification, and focus our attention on the second one, for its simplicity. We are also looking forward to discussing future work on inspecting systems' logs using Quint specifications.

Before describing the specifications, we will give a very short intro to Quint.

2 Quint basics

Quint is a modeling language inspired by TLA+ [1]. Quint's main difference compared to TLA+ is being significantly more in line with standard development practice. While there are arguments that staying away from development practices is beneficial for a specification langauge [1], we find that Quint's focus on integrating standard vocabulary and providing ergonomic tools has been essential for bringing modelling into practice.

Quint is a typed language and it supports basic types such as `bool`, `int`, and `str`, but also structures such as `Records`, `Sets`, `Maps`, `Tuples`, and `Lists`.

Quint's CLI enables user to use its REPL (to experiment with the specification step-by-step), run a simulation of the specification (until an invariant violation is found), or model-check the specification using the Apalache [8] model-checker (and the support for the TLC [9] model-checker is coming soon).

For more details on using Quint, see https://quint-lang.org/.

3 Malachite

Malachite [4] is a Rust implementation of Tendermint consensus algorithm [10]. Being done after the original Golang implementation, it took the lessons from it, one of them being to start with a formal specification.

The specification [5] was written in Quint, and was used in the design phase, as well as for testing the implementation, in two ways.

First, there are deterministic tests of the protocol itself, which are then applied to the implementation. They can be found e.g. in the test `DecideNonProposerTest`.

Second, there are model-based conformance tests, for which Quint generates test steps, and the test requires the state of the implementation to conform with the state of the model. This type of test can be found in the test `test_mbt_part_streaming_random_traces`. (A lot of the model-based testing functionality is accomplished by the application-independent Rust library `itf-rs` [11].)

4 Neutron DEX Liquidity Migration

The migration specification [6] was written for the security audit [12] of the code that was performing migration of users' funds.[1] Alongside manual review, we performed end-to-end tests of the whole process.[2]

Our approach was to create a very light model: the model captures what actions may be taken, but does not model full changes of the state. From the

[1]Independent of the specification for the migration, there is exists another specification of the DEX [13], aiming at checking the protocol rather thant he implementation.

[2]Note that the migration process is non-deterministic, since it needed to be permissionless. Thus, any user could have taken part in the migraiton.

model, we generated a large number of traces. Upon executing each step of those traces, we checked for the step postconditons.

4.1 Detected Problem

The (stripped) scenario that uncovered a problem consisted of three actions: Bob migrates Alice, Charlie migrates Bob, and Alice claims rewards with withdrawal from the new pool. Implicit in that description is that some time passes between the actions.

To understand the problem with this scenario, let us outline the behavior we expected from the system. After Alice's position was migrated from the old pool to the new pool, a certain amount of time passed before Bob's position was migrated to the same pool. During that time, Alice should have earned her new pool rewards. However, it turned out that Alice got the same amount of rewards as Bob (who was migrated at a later point).

Why did that happen? As an implementation detail, there was an internal variable for tracking rewards, that got updated each time a user explicitly claimed rewards. However, that variable was not updated upon initially providing liquidity to the new pool, when rewards were given, too.

5 Future work

Our future work is motivated by a class of problems that we encountered running consensus systems in production: an unexpected behavior happens (for instance, the protocol does not progress, even though it is expected to do so), and we need to understand why. Understanding the *why* for consensus implementations is a difficult problem, and debugging it is almost impossible because of its distributed nature.

Thus, we are currently designing a tool that 1) consumes system logs, 2) transforms them into a (Quint-understandable) trace, and then 3) uses Quint to create the exact state achieved by the trace. Once in the state, we can experiment with the REPL, inspect pre-conditons for expected actions, and compare them to the state of the running system. This approach should allow us to understand potential root causes of the problem faster.

References

1. Lamport, L.: Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002).
2. Quint Language, https://quint-lang.org/, last accessed 2025/08/22.
3. Quint Use Cases, https://quint-lang.org/docs/use-cases, last accessed 2025/08/22.
4. Malachite: Byzantine-fault tolerant consensus engine, https://github.com/circlefin/malachite, last accessed 2025/08/22.

5. Cason, D., Hutle, M., Vanzetto, H., Widder, J.: Malachite Consensus Specifications in Quint, https://github.com/circlefin/malachite/tree/main/specs/consensus/quint, last accessed 2025/08/22.
6. Gavran, I., Ignjatijevic, A.: Model-based Fuzzing for Liquidity Migration, https://github.com/informalsystems/liquidity_migration_test_model, last accessed 2025/08/25.
7. Neutron DEX Documentation, https://docs.neutron.org/developers/modules/dex/overview#overview, last accessed 2025/08/22.
8. Konnov, I., Kukovec, J., Tran, T.-H.: TLA+ model checking made symbolic. Proc. ACM Program. Lang. 3, 1–30 (2019).
9. Yu, Y., Manolios, P., Lamport, L.: Model Checking TLA\({}^\mbox{+}\) Specifications. In: CHARME. pp. 54–66. Springer (1999).
10. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. CoRR. (2018).
11. Ruetschi, R., Biswas, R.: itf-rs, https://github.com/informalsystems/itf-rs, last accessed 2025/08/25.
12. Gavran, I., Ignjatijevic, A.: Neutron Liquidity Migration: Security Audit Report, https://github.com/informalsystems/audits/blob/main/Neutron/2024-03-21_liquidity_migration_audit_report.pdf, last accessed 2025/08/25.
13. Bravo, M., Gavran, I., Moreira, G.: Neutron DEX Specification, https://github.com/informalsystems/neutron-dex-model, last accessed 2025/08/25.