ForMet SG proposal

Summary

We hereby propose the creation of a Formal Methods Sub-governance (ForMet SG) for Free TON.

Mission and Vision

The mission is to facilitate the improvement of the software used on the FreeTON network by using formal verification methods thus dramatically reducing the risk .

The vision is to make FreeTON more secure and robust, and thus to enhance the FreeTON ecosystem and expand the FreeTON community.

Rationale

The world of blockchain and crypto-currencies is a wonderful world for hackers looking for easy targets : smart contracts are small pieces of code managing huge amounts of crypto-currencies, and their automated behavior and immutability make them easy targets. The short history of blockchains is already full of stories of millions of USD stolen or lost in many smart contracts, starting from the DAO and ERC20 awful stories. Most of these contracts had been tested by developers (using frameworks like Truffle, etc.) and many of them were audited by “security experts”. Despite these efforts, there were still bugs to be exploited in them. This is due to the limited nature of such verification techniques.
On the other hand, formal methods provide much more bullet-proof verification techniques: they show that a smart contract code satisfies a formal specification, not for a few tested cases, but for all possible executions. This is a very strong property to get, that gives much more trust in the smart contracts than just testing or auditing. Yet, formal methods are limited by the expertise and skills required to deploy them on real life smart contracts.
The goal of this new sub-governance is to build a team of strong experts from various companies and to apply their expertise in formal methods to secure the Free TON ecosystem and its most important smart contracts.

Scope of Activities

The proposed sub-governance shall perform the following activities:

- Organization and running of formal verification contests
- Distribution of funds to contest winners
- Developing and improving the Free TON community guidelines on formal verification

- Facilitating the development and application of open source tooling for formal verification purposes
- Encouraging the creation of further formal verification teams

Formal verification is intended to be applied to the following software:
- Smart contracts (especially those essential to the Free TON infrastructure)
- TON compiler and other essential software
- Other software related to the Free TON blockchain with critically important behaviour

Subgovernance Organization and Management

Membership

Initial Members

Name	**Telegram**	**Background**	**Public Key**
Fabrice LE FESSANT	Telegram	Former Researcher in Prog. Lang. and Distr. Systems, CEO at Origin Labs, CTO at OCamlPro	4aca372ed9695ab42cc8ba7fd7f56d11c2401611c2d513bbc28beb5c7f4363a1
Sergey EGOROV	Telegram	Software manager/director with ~20 years of experience, Cofounder of Pruvendo	67dd20b9a760ae538a7f24ebfbaaf09a7075b4617a7ad09c19503c2551f57d81
Thomas SIBUT-PINOTE	Telegram	PhD. in Formal Proof (https://www.theses.fr/2017SACLX086) using Coq. Engineer at OCamlPro/Origin Labs. Initial author of the Tezos Michelson formal proof framework in Coq and F*.	50384ec36bee19914526f436a0adf57d0c35389934b5aaca15db5b5e89f42aa0
Evgeniy SHISHKIN	Telegram	Researcher in the field of Formal Methods in Software. Pursuing PhD in Institute of Systems Programming (ISP RAS). Senior Researcher at InfoTeCS R&D (software security/reliability group). See my homepage.	6ff61c1a7bb09795f7b5d5514dd710efb72e9557654d362ef208fde545ba7a33
Sergey TYURIN	Telegram	Programmer 30+ years of experience	2c0ec55a109eb466d9db5ee7c3adb075e77627ade83ae17cea847671ab8f0a85

Andrey LYASHIN	Telegram	Entrepreneur. Mathematician, Coq-programmer with 10+ years of experience. Co-founder of Pruvendo and consensusresearch.org.	cec27f6cfdadadc5da135875d5988019bd8a760fe6e16fe1f49459cf6d18f9e7
Nickolay VASILEV	Telegram	Formal Verification Engineer, 8+ years of experience in Coq	b722871cc121019663e982eaf24958b2ca9ffad4472d8d0a98110d5eb5082004
Boris IVANOVSKY	Telegram	25 years in system programming, half of them in compiler dev. Product owner of TON dev tools as a job, doing smart-contracts for FreeTON as a hobby.	1a99622e54b4e87d603dd87c9cc936b388b2a0e1979bb56d4039cfad0fbadc8c
Mitja GOROSHEVSKY	Telegram	CTO of TONLabs, the leading ideologist of FreeTON	6ff322ad669dfad2f396b98bdc8690cc49926f6a10cd7f10d07f031841cf09ef

The initial members shall oversee the efforts related to building the SG. They will also act as jurors until the SG jury is assembled.

Member Admission

Membership is open to everyone with proven experience in the field of formal verification, provided that a simple majority of the SG members approves the prospective member via on-chain voting during the specific contests being run at least twice a year. The election shall be made by a simple majority.

Membership Termination / Dismissal

The Membership in the SG can be terminated at any time by submitting a special on-chain proposal that must be supported at least by ⅔ of SG members. In case of voluntary resignation, the resigning member is supposed to initiate the proposal by herself, or, in case of impossibility to do so, such a proposal can be initiated by any SG member.

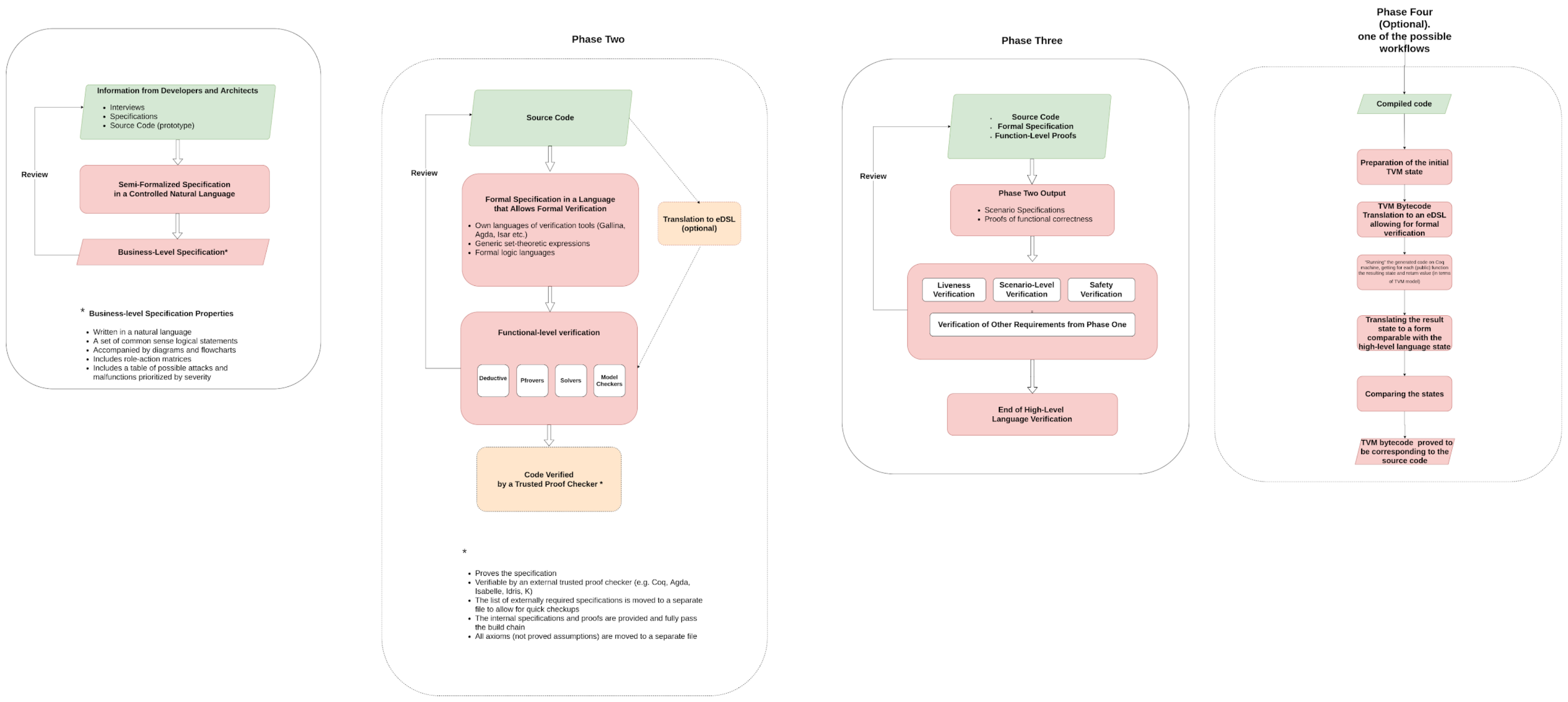
Communication

The communication on the matters related to the activities of the SG shall happen through the following channels:

- Website on gov.freeton.org (publication of contest proposals, information on membership etc).
- Forum on gov.freeton.org
- Telegram group (open membership)
- Regular (weekly/biweekly) call

Formal Verification Workflow

We hereby propose a workflow for formal verification. It comprises four phases, and each of them can be the subject of a separate contest. It’s illustrated at the next page and discussed further later.


Information from Developers and Architects
Interviews
Specifications
Source Code (prototype)
Review
Semi-Formalized Specification in a Controlled Natural Language
Business-Level Specification*
Phase Two
Source Code
Review
Formal Specification in a Language that Allows Formal Verification
Own languages of verification tools (Gallina, Agda, Isar etc.)
Generic set-theoretic expressions
Formal logic languages
Translation to eDSL (optional)
Functional-level verification
Deductive
PProvers
Solvers
Model Checkers
Code Verified by a Trusted Proof Checker *
Phase Three
Source Code
Formal Specification
Function-Level Proofs
Review
Phase Two Output
Scenario Specifications
Proofs of functional correctness
Liveness Verification
Scenario-Level Verification
Safety Verification
Verification of Other Requirements from Phase One
End of High-Level Language Verification
Phase Four (Optional). one of the possible workflows
Compiled code
Preparation of the initial TVM state
TVM Bytecode Translation to an eDSL allowing for formal verification
"Running" the generated code on Coq machine, getting for each (public) function the resulting state and return value (in terms of TVM model)
Translating the result state to a form comparable with the high-level language state
Comparing the states
TVM bytecode proved to be corresponding to the source code

* **Business-level Specification Properties**

- Written in a natural language
- A set of common sense logical statements
- Accompanied by diagrams and flowcharts
- Includes role-action matrices
- Includes a table of possible attacks and malfunctions prioritized by severity

*

- Proves the specification
- Verifiable by an external trusted proof checker (e.g. Coq, Agda, Isabelle, Idris, K)
- The list of externally required specifications is moved to a separate file to allow for quick checkups
- The internal specifications and proofs are provided and fully pass the build chain
- All axioms (not proved assumptions) are moved to a separate file

Phase 1

1. On the basis of information obtained from smart contract developers and architects (interviews, specifications, and source code), a semi-formalized specification in a controlled natural language is created.
2. On the basis of the semi-formalized specification, a business-level specification is created so that:
    a. It is written in a natural language
    b. It contains a set of common-sense logical statements
    c. It is accompanied by diagrams and flowcharts
    d. It includes role-action matrices (optional)
    e. It includes a table of possible attacks and malfunctions prioritized by severity
3. The business-level specification shall be reviewed and approved in due course by smart contract developers and architects.

Phase 2

1. On the basis of smart contract source code, a formal specification should be provided in a language that allows smart verification to be formally described. This language can be either the own language of the verification tool (proof assistant, Prolog etc.) being used or be a specific formal logic language such as:
    a. First-Order Logic
    b. Temporal logic of different kinds
    c. Separation Logic
    d. TLA+ family languages
    e. Reachability Logic
    f. Event-B
    g. …

    If need be, the source code can be translated to an embedded domain-specific language (eDSL).
2. Using the formal specification, the functional verification is performed in one of the following ways:
    a. Semi-mechanical deductive reasoning (Coq, Agda, etc)
    b. Automated theorem provers (Vampire, etc)
    c. SAT/SMT solvers (Z3, AltErgo)
    d. Symbolic/Explicit Model-Checkers
3. After review against the source code, the outcome is the code verified by a trusted proof checker so that:
    a. It proves the specification
    b. It is verifiable by an external trusted proof checker (e.g. Coq, Agda, Isabelle, Idris, K, whyML, Z3, AltErgo)

    c. The list of externally required specifications is kept separately for the sake of quick checkups
    d. The internal specifications and proofs are provided and fully pass the build chain
    e. All axioms (not proved assumptions) are kept in a separate file

Phase 3

1. On the basis of the source code, the formal specification and functional proofs, the following artifacts are created:
    a. Scenario specifications
    b. Proofs of functional correctness
2. The following verification is performed:
    a. Scenario-level verification
    b. Gas consumption verification
    c. Safety verification
    d. Liveness verification
    e. Verification of other requirements from Phase 1

Phase 4 (optional)

The goal of Phase 4 is to verify that the TVM code generated for the particular contract works identically to the source code verified at the former stages. Thus, the compiler will be verified but for the particular contract only. The kind of verification can be achieved, for example, by the following workflow:

1. TVM bytecode is translated to an eDSL allowing for formal verification if needed
2. Mapping between high-level state and TVM-level stated must be described
3. The eDSL code is run
4. The resulting state is translated to a form comparable with the initial state
5. The states are compared, and the TVM bytecode is proved identical to the source code

Formal Verification Contest Organization

The SG shall follow the established Free TON practices when organizing contests.

The Jury

Contests are managed by the elected Jury. At the inception stage of the SG, the Initial Members shall organize the Jury Selection Contest; before the Jury is elected, the Initial Members shall act as jurors.

Selection Criteria

The Jury is elected from members with proven expertise in formal verification. Further requirements shall be described separately in the proposal for the Jury selection contest.

Some samples of Jury selection contest proposals for reference:

- https://forum.freeton.org/t/free-ton-web-design-jury-selection-contest-1-0/4511
- https://forum.freeton.org/t/contest-global-community-subgov-jury-selection/6914
- https://blog.freeton.org/en/analytics-support-jury-selection-contest-results/

Functions of the Jury

The Jury shall perform the following functions:

- Ensuring the smooth operation of the SG
- Developing and reviewing contest proposals (CPs)
- Accepting and reviewing contest proposals
- Assessing contest submissions from participants
- Choosing contest winners
- Distributing funds between contest winners

Term of office of the Jury

Jurors are elected indefinitely; if a juror leaves the Jury, a new by-election contest shall be held by creating a proposal and voting on it.

Remuneration to the Jury

The matter is covered in the following section. Further details shall be included in specific contest proposals as they are on a contest-by-contest basis and described in each contest, because depending on the contest, terms may vary.

Budget and Remuneration

The contest budget

The budget distribution for each type of the contest is described in the table below. It’s important to note that this budget is proposed for **each** contest, not for the whole bunch of them.

Contest	1st place prize	Total budget	Details

Smart Contract Verification Contests			As described above these contests are splitted into the phases and each stage is judged separately
Phase 1	50 kTON	100 kTON	
Phase 2	350 kTON	900 kTON	
Phase 3	350 kTON	900 kTON	
Phase 4	350 kTON	900 kTON	This phase is optional as discussed above
Open Source Tooling Contest	500 kTON	1 MTON	Tooling to improve the quality of smart contracts in the ecosystem
TVM Verification Contests	500 kTON	1 MTON	2 contests planned: TVM formal definition and program logic for TVM bytecode
Bug Bounty Contest	500 kTON	550 kTON (50 kTON for jury, the contest is opened for each appropriate bug claim)	Incentives for bug finders. As the bugs are not expected for the formally verified code it should be a big bounty for chasers who dare to question the overall technology and approaches used by ForMet teams
TVM Audit Contest	300 kTON	700 kTON	Audit of the TVM Execution Engine in Rust prior to later formal verification
Specification contest	500 kTON	1 MTON	Contests for those parts of the system that are not planned to be formally verified in the near-term but still useful for the audit and other activities such as blockchain model, node model, consensus etc.

Bug Bounty

This contest opened only when somebody claims he/she found a serious bug inside the code that is claimed as verified and at least one jury member believes it’s a true claim. Only one submission per contest that can be either accepted or rejected. If the claim is finally accepted it’s considered as an exception and some explanations and plans for improvement are expected from the team.

Jury Bounty

Normally, the jury bounty is 5% of the overall budget, but for phases 2, 3 and 4 of Smart Contract Verification Contests it’s planned to raise it to 20% as it requires extremely serious efforts.

This amount will be distributed among jurors who vote and provide feedback. This percentage will be awarded on the following basis:

The percentage of tokens awarded to the jury will be distributed based on the number of votes each juror casts. For example, if one juror votes 20 times and another juror votes 5 times, the juror who votes 20 times will get 4 times more tokens than the juror who votes 5 times. Detailed feedback is mandatory in order to collect any rewards.

Runner-up places award distribution

The award distribution for the places lower than the first will be defined inside the proposal for each contest and discussed and accepted inside the subgov, but normally, the total award for lower places should roughly exceed the 1st place award, staying inside the overall contest budget.

The global budget

The global budget of the ForMet SubGov for 2021 is split into the six stages, where each stage implies contributions approximately twice per quarter. The budgeting for each next stage is approximate and may be corrected. It should be provided upon the specific inquiryto the main governance and through the regular process of approval. The plan for each stage (including the first one) is the best guess so the subgov may add, modify or remove some articles without looking for additional approval. However, before asking for the next asking of contributions, the subgov must report both about all the finished contests as well as about deferred or canceled, also providing the information about the remaining balance that has to be taken into account while counting the amount of tokens to be provided for the next stage.

The contributions for each next stage is asked when all contests for the previous stage are completed and approved by the jury. In case some of the contests require more time or are canceled, the subgov has a right to issue an on-chain voted proposal to move it to the next stage. This decision must be explicitly noted in the request for the next stage of contributions.

In case the particular contest is long and takes more than one stage, the contributions for it will be asked only at the last stage. All the participants are supposed to understand and accept the risk of rejecting the corresponding financing.

The Bug Bounty contest is assumed for each second stage but if it doesn't happen (as supposed), its budget is automatically transferred to the next second stage without the additional request from the main governance.

Contest type	Quantity	Budget (TON)	Details
Stage Ia, Q2CY21			
Phase 1	2	200 000	SMV, Elector
Phase 2	1	900 000	TIP-3
Specification Contest	1	1 000 000	To be discussed, but the Solidity formal interpreter is the best candidate
Total (Stage 1a)			**1 100 000**
Stage Ib, Q2CY21			
Phase 1	1	100 000	PBTG (or another contract)
Phase 2	1	900 000	SMV
Open Source Tooling	1	1 000 000	To be discussed
Bug Bounty Contest	1	550 000	One contest is budgeted for each stage while it's hoped this contest never happens
Specification Contest	1	1 000 000	To be discussed, but the Solidity formal interpreter is the best candidate
Total (Stage 1b)			**3 550 000**
Stage IIa, Q3CY21			
Phase 1	1	100 000	Multisig 2
Phase 2	1	900 000	Elector
Phase 3	1	900 000	TIP-3
Total (Stage 2a)			**1 900 000**
Stage IIb, Q3CY21			
Phase 1	1	100 000	DePool 2

Phase 2	1	900 000	PBTG
Phase 3	1	900 000	SMV
Open Source Tooling	1	1 000 000	To be discussed
TVM Verification Contests	1	1 000 000	TVM formal definition
Bug Bounty Contest	1	550 000	One contest is budgeted for each stage while it’s hoped this contest never happens
Specification Contest	1	1 000 000	To be discussed, but the blockchain model is the best candidate
Total (Stage 2b)			**5 450 000**
Stage IIIa, Q4CY21			
Phase 1	2	200 000	Some new contracts
Phase 2	1	900 000	Multisig 2
Phase 3	1	900 000	Elector
Phase 4	1	900 000	TIP-3
Total (Stage IIIa)			**2 900 000**
Stage IIIb, Q4CY21			
Phase 1	2	200 000	Some new contracts
Phase 2	1	900 000	DePool 2
Phase 3	1	900 000	PBTG
Phase 4	1	900 000	SMV
Open Source Tooling	1	1 000 000	To be discussed
TVM Verification Contests	1	1 000 000	Program logic for TVM bytecode
Bug Bounty Contest	1	550 000	One contest is budgeted for each stage, while it’s hoped this contest never happens
TVM Audit Contest	1	700 000	
Specification Contest	1	1 000 000	To be discussed, but the node

			model is the best candidate
Total (Stage IIIb)			**7 150 000**

For the CY22 and upcoming years, another budgeting shall be conducted.

Subgov wallet

The subgov wallet is placed upon the experimental SMV contract (developed by RSquad - the winner of the corresponding contest) with the address `0:1ec958fd022ab1d479dd722283fe5fd1d9de7196ee7f09f96b68e435776548c1`.

First stage investment

If the present proposal is accepted by the main governance, the transfer of **1 100 000** TON is expected to the address `0:1ec958fd022ab1d479dd722283fe5fd1d9de7196ee7f09f96b68e435776548c1`. As mentioned above, the address is owned by the SMV smart contract, and its security is checked by the whole team.