Formal Verification of the Tezos Codebase

Formal Methods at Nomadic Labs

Zaynah Dargaye

Journée GT MFS

nomadic labs

At Tezos Origin

Tezos is implemented in OCaml, a powerful functional programming language offering speed, an unambiguous syntax and semantic, and an ecosystem making Tezos a good candidate for formal proofs of correctness.

Tezos White Paper – L.M.Goodman 2014.

Nomadic Labs Expertise

A R&D team with a strong background in formal specification and verification, as well as in programming language theory.

In this talk we will focus only on the OCaml implementation of Tezos from `gitlab.com/tezos/tezos` (Octez)

Daily Usage of Lightweight Formal Methods

Informal Specification

Documentation at `http://tezos.gitlab.io/` maintained with the codebase on git. Readme files for module, and annotated api files, in progress.

Type System

OCaml type system guarantees.

Testing

Classical unit/integration testing, Property-based testing, system testing, and testnet.

Code Reviews

Every update to the codebase is reviewed by several experimented engineers.

Michelson

The *assembly* smart contract language of Tezos, low-level stack-based, strongly typed without jump instructions.

Mi-Cho-Coq

Michelson deep-embedding in Coq, with a weakest precondition interpreter enabling functional verification.
Functional verification of a Decentralized Exchange, Dexter.

Ph.D. Static Analysis for Michelson

Abstract Interpretation of Michelson Smart Contract – Guillaume Bau
advisors: A. Miné LIP6, Nomadic Labs mentors: V. Botbol and M. Bouaziz.

nomadic labs

We use the HACL* implementation, developed and certified in F*.

Useful Library

A comprehensive collection of crypto primitives under a unified API , providing agility (an API for several algorithms), and multiplexing (several implementation choices for an algorithm).

High Performance

Hand-tuned C+asm for x64, with fallback high performance C version for several primitives.

Formally Verification Guarantees

safety (memory safety, no calls to illegal operations), functional correctness, secret independence (resistance to timing- and cache-based side channel attacks).

Section 1

Daily Usage of Formal Methods, Under Construction

A Cutting-Edge Technology

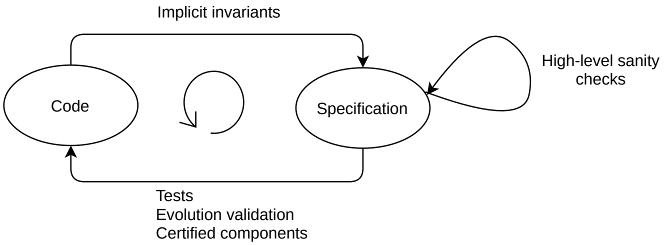
High shipping frequency, evolving requirements, evolving priorities, inter-dependent features in terms of both impacted code components and required expertise.

High Quality Technology

Critical data managements, adversarial deployment contexts, asset and value of the brand.

Having both means having a development process that includes formal methods.

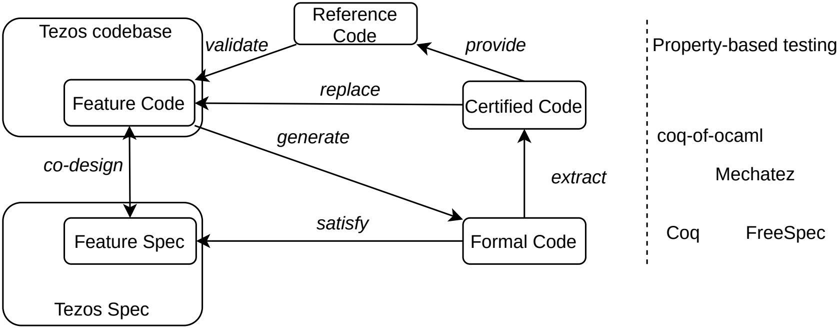
Co-design
the specification and the code
focusing on prioritized developments.



Section 2

Efficient Program Proof

Formal Tools for the Verification Process



Code Specification: the Reference Code

FreeSpec "A framework for implementing and certifying impure computations in Coq"

Code Semantics

Extend the program with effects framework with a contract-based reasoning.

Program Proof

Hiding sophisticated theories and administrative proof details.

Modular

Reasoning on one component at the time, and component compositions.

Trust Bases and Maintainability

Based on well-established theoretical features, open-source and developed at ANSSI and IRIF by Thomas Letan and Yann Regis-Gianas, both at Nomadic Labs now.

Certified Code Injection

Getting the FreeSpec interface, easing injection of extracted code

MECHATEZ:

- based on COQFFI developped by T. Letan and in coq-community,
- integrated into Tezos codebase and generated at building time,
- objective is to give the COQ API of the whole codebase in order to focus on verifying a particular component.
- also provide injection of extracted code features.

Can we validate the observational equivalence *btw.* an OCaml function f_o and an extracted one f_c when the precondition P is satisfied by the pre-state Pre:

$$\forall Pre, P(Pre) \Rightarrow f_c \approx f_o?$$

Property-based Testing

Testing predicate satisfaction by a function in some generated pre-states.

$P(Pre)$ Generation

Predicates on the input state, we need simple state constructors.

Observational Equivalence $\approx$

Comparing *probes*: state observable data (alloc, set, and free).

$$\forall Pre, Pre \in \texttt{stategen}\ P \Rightarrow \texttt{probes}\ f_c = \texttt{probes}\ f_o \Rightarrow f_c \approx f_o.$$

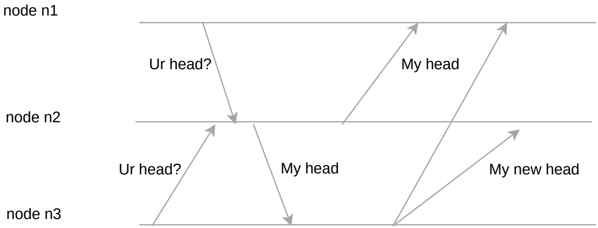
nomadic labs

Section 3

Verifying Correct Integration in the Codebase

Blockchain Protocols at Glance

A Message passing protocol enabling a network to manage a replicated append-only data structure (*blockchain*).



Blockchain Properties

Eventual Consistency

Every node will eventually share the same view of the blockchain.

Common Prefix

Every node shares a common prefix of this shared common view.

Chain Growth

As long as new block are appended, this common prefix grows.

Blockchain Content

Writing entries in the ledger are consistent with its previous content.

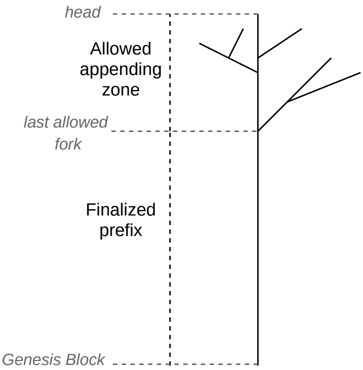
(BBP,Garay&al.) (BCADT,Anceaume&al.)

nomadic labs

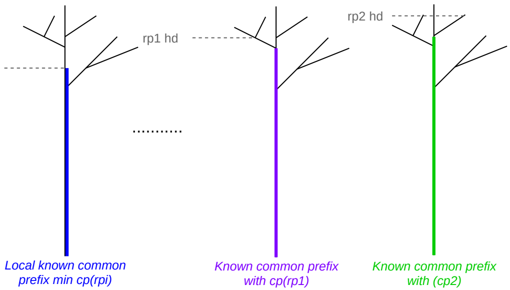
Node blocktree

local chaining properties:

- block chaining,
- Chain integrity,
- In chaining allowed window, (latency absorption)
- Agreement reaching,
- Potential head update.



Node Properties



Node common prefix:

- peer common prefix with each remote peer,
- known common prefix with its remote peers.

Node chain growth: The node learns a new head from at least one of its remote peers.

- Common Prefix: lift (extension) of every node's known common prefix,
- Chain Growth: New block injection ensures that at least one node will eventually have a new head to advertise.

In a suitable execution model: Communication model, Scheduling and fairness.

How to model Faulty, Byzantine and Rational nodes?

Section 4

A Blockchain Protocol with a Self-amendment Mechanism

Staying a cutting-edge blockchain with hard fork control thanks to self-amendment.

Self-amendment Consequences

The economic protocol (EP): ledger entries, agreement and self-amendment.
A *score*: abstraction for agreement reaching and chain validity.
The shell, the structural node, uses *score* to drive block chaining and head switches.

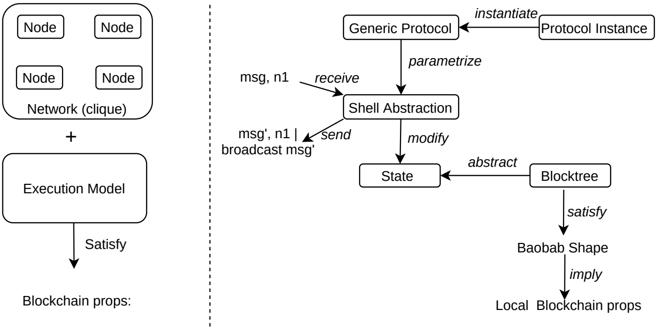
Impact on Blockchain Properties

Abstract EP $\simeq$ EP API $+$ score properties $+$ content appending validation.

Each EP is an instance of an abstract EP.

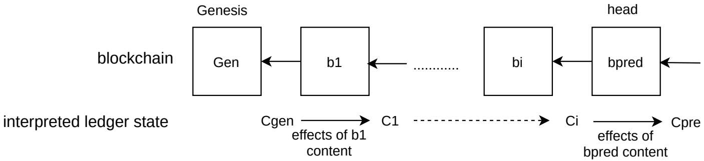
High-level Specification and Properties

COQUILLE a formal framework for verifying component integration in the Tezos codebase.



nomadic labs

Blockchain Content Validity



Legder Abstraction

The ledger content of a chain (whose head is) *ch* is called a context: C. A context is valid: $\mathcal{V}\ C$ if it satisfies the economic protocol invariants. The genesis context is valid.

By Induction/Construction on Application

Assuming $\mathcal{V}\ C_{pre}$, a block candidate *b* appending will succeed if:
bpred is the predecessor of *b*, and *b* content effects on C_{pre} produces C_{post} such as they maintain the economic protocol invariants.

Section 5

Classical 3-layer Verification in Industrial Setting

At a Glance

Formal Consistency

Compliance of a component specification with the global specification in Coquille

Generating FreeSpec program

Generation from the component implementation of a FreeSpec program

Program Proof

Verification in FreeSpec with dedicated to our code tactics

Extracting a Reference Implementation

Thanks to Mechatez, extracting a reference implementation of the component that can be integrated in the codebase.

Validating the Implementation by Model-based Testing

Observational equivalence between the implementation and the reference one.

Current Status

Work in progress, modularity enables incremental developments.
A first use case on the road in the economic protocol side.
Concurrent or distributed codes and economic protocol switching are out of the scope.

Ph.D. Thesis for Communicating Components

Formal Specification and Verification of Message Passing Protocol – Paul Laforgue,

advisors: G.Castagna, and G.Bernardi IRIF, Nomadic Labs mentors: Z. Dargaye, T. Letan and Y. Regis-Gianas.