An Open Technical Assessment for Formal Methods in Frontend Engineering

A Two-Phase Challenge in Operational Semantics and TypeScript Implementation

ab@normacore.dev

Revision 1.0 April 8, 2026

Abstract

We present an open technical assessment designed to evaluate a candidate's ability to reason formally about program state, construct verified operational semantics, and translate mathematical models into production-quality web software. The assessment is structured in two phases: (1) a formal specification and verification exercise in Lean 4, and (2) a faithful TypeScript implementation paired with a browser-based user interface. The subject of both phases is a conforming interpreter for the Brainfuck programming language under a fixed set of domain constraints. Evaluation criteria weight architectural understanding and proof correctness alongside code quality, with the expectation of a substantive follow-up technical conversation on every submitted design decision.

Contents

---

1 Introduction

This document describes an open engineering challenge distributed via X and LinkedIn. It is intended for candidates with interest in formal methods, programming language theory, and frontend systems engineering.

The challenge is structured around a deliberately minimal subject domain — the Brainfuck programming language — whose simplicity makes it tractable for formal verification while still admitting non-trivial design decisions around type representation, error modeling, and loop semantics. Candidates are expected to make independent architectural choices at every level; no skeletons or scaffolding are provided.

> *Note: Candidates are welcome to use AI-assisted tools during development. Submissions are evaluated primarily on architectural understanding. Every design decision is subject to a follow-up technical discussion.*

2 Domain Constraints

Both phases of the assessment operate under the following fixed domain constraints.

Constraint 1 (Tape)**.** The interpreter tape consists of exactly 30,000 cells, indexed 0 through 29,999.

Constraint 2 (Cell Type)**.** Each cell holds an 8-bit unsigned integer value in the range $[0, 255]$.

Constraint 3 (Arithmetic Wrapping)**.** Cell arithmetic is performed modulo 256. Specifically, decrementing 0 yields 255 and incrementing 255 yields 0.

Constraint 4 (Pointer Bounds)**.** Pointer movement below index 0 or above index 29,999 must produce a well-typed error state. The representation of this error state is left to the candidate.

3 Phase I: Formal Modeling in Lean 4

3.1 Overview

Candidates must define a complete formal model of the Brainfuck interpreter from scratch in Lean 4. The deliverable for this phase is a Lake project located in a `lean/` directory at the repository root, buildable via `lake build` with no errors or warnings.

Candidates may use Lean's standard library and Mathlib. The choice of whether and how to use Mathlib is itself part of the evaluation: reaching for `Fin 30000` or `Vector` to encode invariants at the type level reflects a qualitatively different understanding than patching runtime checks onto untyped structures.

3.2 Required Definitions

Requirement 1 (Instruction Set)**.** Define the complete Brainfuck instruction set as an inductive type.

Requirement 2 (Interpreter State)**.** Define the interpreter state as a structure. The choice of field types is the candidate's primary design decision. Candidates should be prepared to justify each type choice — in particular, how pointer bounds and tape size are represented.

Requirement 3 (Step Function)**.** Define a `step` function capturing the operational semantics of each instruction as a state transition.

Requirement 4 (Result Type)**.** Define a `Result` type that formally models all possible interpreter outcomes, including at minimum: normal termination and out-of-bounds pointer error. Additional failure modes are at the candidate's discretion.

3.3 Specifications and Proofs

After the definitions, candidates must state the properties their model satisfies as typed Lean propositions — `theorem` or `lemma` declarations with full type signatures. Propositions are evaluated independently of their proofs: precision and provability are assessed on their own terms.

All stated propositions must then be proved. Proofs must compile cleanly under `lake build`. No `sorry` placeholders and no axioms beyond Lean's standard library and Mathlib are permitted.

4 Phase II: TypeScript Implementation and User Interface

4.1 Core Engine

Candidates must implement the Brainfuck interpreter in TypeScript such that it structurally mirrors the formal model defined in Phase I: the same types, the same separation of concerns, the same failure modes. State must not be mutated in place; the engine must consist of pure functions returning new state objects.

4.2 Parsing and Evaluation

Parsing and evaluation must be treated as explicitly separate phases. Bracket matching and program validation must occur prior to execution, producing a structured program representation that the evaluator then consumes. The representation is the candidate's design decision.

4.3 Execution Model

The interpreter must not block the main browser thread. Long-running programs and non-terminating loops must not freeze the user interface. The candidate's chosen approach to non-blocking execution must be documented in the `README.md` and justified.

4.4 User Interface

The design of the user interface, repository structure, and frontend toolchain are entirely at the candidate's discretion. What is evaluated is whether the architectural constraints described above are satisfied and whether the candidate can defend their choices.

4.5 Correctness Requirement

The interpreter must produce correct output for the standard Hello World Brainfuck program. Additional programs will be used for correctness testing during review.

5 Submission

Completed submissions should be sent to **ab@normacore.dev** with the subject line:

```
Brainfuck Frontend Challenge -- [Your Name]
```

Each submission must include:

1. A link to a public Git repository containing both phases.
2. A link to the candidate's resume or LinkedIn profile.

We read every submission carefully.