Project Prometheus v0.4 Work Plan: The Autonomous Mathematician

Generated by Gemini

July 17, 2025

Abstract

This document outlines the detailed work plan for the next major phase of the Project Prometheus demonstrator, v0.4. This phase, titled "The Autonomous Mathematician," is designed to transition the system from demonstrating basic capabilities to tackling genuinely complex problems that require multi-step, strategic reasoning. Where v0.3 proved a single, simple theorem, v0.4 will implement a sophisticated reasoning engine capable of navigating complex proof searches, learning from its mistakes, and beginning to exhibit the kind of abstract reasoning that is a core requirement for ultraintelligence as defined in the project's foundational documents.

---

1 Task 1: Implement a Multi-Step Reasoning Engine

1.1 Rationale

As per the roadmap (Phase 3), the system must move beyond simple, single-shot tasks and demonstrate performance in a complex, formal domain. Real mathematical theorem proving is not a single action but a sequence of logical steps. This task upgrades the agent from a simple "prover" to a "reasoner" that can build a proof step-by-step.

1.2 Methodology

1. **Proof State Management:** Implement a `ProofState` class within the `LeanTool`. This class will be responsible for parsing the output from the Lean server to track the current goals, hypotheses, and the overall state of the proof.
2. **Tactic Generation in `CoderAgent`:** The `CoderAgent`'s `prove` method will be refactored. Instead of attempting to generate a complete proof, it will now be a `generate_tactic` method that takes a `ProofState` as input and generates the single, most promising Lean tactic to apply next.
3. **Iterative Proof Loop in `MCSSupervisor`:** The `MCSSupervisor` will be upgraded with a `run_proof_cycle` method. This method will manage an iterative loop:
   (a) Initialize the proof with the theorem and get the initial `ProofState` from the `LeanTool`.
   (b) Pass the current `ProofState` to the `CoderAgent` to generate the next tactic.
   (c) Apply the generated tactic using the `LeanTool`.
   (d) Repeat the cycle until the `LeanTool` reports "goals accomplished" or a specified step limit is reached.

1.3 Deliverables

- An updated `LeanTool` with a `ProofState` management class.
- An updated `CoderAgent` with a `generate_tactic` method for iterative reasoning.
- A new `run_proof_cycle` method in the `MCSSupervisor` to orchestrate the multi-step proof process.

1.4 Verification

The system can successfully prove a theorem that requires at least three sequential, distinct tactics in Lean, demonstrating its ability to build a coherent line of reasoning.

2 Task 2: Implement a "Proof Tree" Search Strategy

2.1 Rationale

A linear, step-by-step reasoning process will inevitably get stuck on complex problems. To tackle a "grand challenge" (Phase 3), the agent must be able to explore different lines of reasoning, backtrack from dead ends, and strategically choose the most promising path. This task implements a "Tree-of-Thought" search mechanism, a crucial step towards more robust problem-solving.

2.2 Methodology

1. **Proof Tree Data Structure:** Implement a `ProofTree` class. Each node in the tree will represent a `ProofState`, and the edges will represent the tactics applied. This creates a complete, machine-readable history of the agent's reasoning process.
2. **Heuristic Search in `PlannerAgent`:** The `PlannerAgent` will be upgraded to act as a "search strategist." Its prompt will be enhanced to take the entire `ProofTree` as input. Its task will be to analyze the tree and identify the most promising node (i.e., the most promising partial proof) to expand next. This introduces a layer of meta-cognition, where the agent reflects on its own reasoning process.
3. **Update `MCSSupervisor` for Tree Search:** The `run_proof_cycle` will be updated to manage the tree search. It will build the `ProofTree` and, at each step, consult the `PlannerAgent` to decide which branch to explore. If a tactic results in an error, that branch is marked as "failed," and the supervisor will ask the `PlannerAgent` to choose a different, more promising branch.

2.3 Deliverables

- A `ProofTree` class for managing the search space.
- An updated `PlannerAgent` that can analyze the `ProofTree` and guide the search.
- An `MCSSupervisor` capable of managing the tree search, including backtracking from failed proof attempts.

2.4 Verification

The agent can successfully solve a theorem where a simple, linear ("greedy") application of tactics would fail. The logs must clearly show the agent exploring one branch, hitting a dead end, backtracking, and then successfully finding the proof along a different branch.

3 Task 3: Refine Meta-Learning - Learning from Failed Proofs

3.1 Rationale

This task directly implements the "Refine Meta-Learning capabilities" goal from the roadmap (Phase 3, Task 4). A truly intelligent agent should not just backtrack from failure; it should *learn* from it. This task gives the system a "memory" of its mistakes, allowing it to refine its own proof strategies over time.

3.2 Methodology

1. **Failure Analysis in `EvaluatorAgent`:** The `EvaluatorAgent` will be given a new role in the theorem-proving process. When a proof branch fails, the `EvaluatorAgent` will receive the failed sequence of tactics and the final Lean error message. It will then use the LLM to generate a concise, natural-language "critique" of the failed strategy.

- **Example Critique:** "The agent attempted to use a rewrite tactic (`rw`) before the necessary preconditions were met. The `induction` tactic should have been used first to break the problem into base and inductive cases."

2. **Update `PlannerAgent` with Contextual Memory:** The `PlannerAgent`'s prompt for guiding the tree search will be updated to include a list of the critiques from all previously failed branches. This provides the agent with a contextual "memory," allowing it to avoid repeating the same category of mistake.

3.3 Deliverables

- An updated `EvaluatorAgent` with a `critique_failed_proof` method.
- An updated `PlannerAgent` that incorporates this historical feedback into its search strategy.

3.4 Verification

After failing to prove a theorem using one strategy, the system can use the generated critique to successfully prove the same theorem on a subsequent attempt, and the logs show that it explicitly avoids the previously failed line of reasoning.