A Formal Mathematical Framework for an AI Agent Integrating Ontologies, Bayesian Inference, Kalman Filters, Large Language Models, Token-Aware Monte Carlo Tree Search, and Homotopy Type Theory

Juan Zambrano - Marius Dinus

October 10, 2024

Abstract

This paper presents a formal mathematical framework for an AI agent designed to perform continuous navigation tasks within a structured environment. The agent leverages a combination of ontologies, Bayesian updating, Kalman filters, Large Language Models (LLMs), Token-Aware Monte Carlo Tree Search (MCTS), and Homotopy Type Theory (HoTT). The framework is meticulously defined through precise mathematical constructs, algorithms, and formalizations suitable for both theoretical analysis and practical implementation. Additionally, the integration of the Coq proof assistant within a Docker environment is detailed to facilitate computational verification of HoTT-based hypotheses, ensuring the agent's reasoning processes are both robust and verifiable.

Contents

1 Introduction

The development of autonomous AI agents capable of navigating complex environments necessitates the integration of diverse computational and mathematical frameworks. This paper introduces a comprehensive framework that synthesizes ontologies, Bayesian inference, Kalman filters, Large Language Models (LLMs), Token-Aware Monte Carlo Tree Search (MCTS), and Homotopy Type Theory (HoTT). The agent operates within a continuous two-dimensional (2D) environment, leveraging these components to make informed decisions, reason about its surroundings, and verify its actions through formal proofs using the Coq proof assistant within a Dockerized setup.

1.1 Motivation

Autonomous agents must navigate environments efficiently while reasoning about their actions and the underlying structures governing their interactions. Traditional navigation algorithms often lack the capacity for deep reasoning and formal verification, which are crucial for tasks requiring high reliability and adherence to complex rules. By integrating advanced mathematical theories and probabilistic models, this framework aims to enhance the agent's decision-making capabilities and ensure the correctness of its actions.

1.2 Contributions

This paper makes the following key contributions:

1. **Ontology Integration**: Formalizes the representation and navigation of knowledge structures using directed graphs.
2. **Probabilistic Inference**: Implements Bayesian updating and Kalman filters to maintain and update the agent's belief states.
3. **Decision-Making Mechanism**: Employs Token-Aware Monte Carlo Tree Search (MCTS) to balance exploration and exploitation within token budget constraints.
4. **Reasoning and Verification**: Utilizes Large Language Models (LLMs) for generating reasoning traces and integrates Homotopy Type Theory (HoTT) for formal verification of hypotheses via the Coq proof assistant.
5. **System Integration**: Details the Docker-based environment setup for seamless interaction between the agent's components and the Coq verifier.

2 Universal Notation

To ensure clarity and consistency throughout this document, we establish the following universal notation:

- **Scalars**: Lowercase letters $a, b, c \in \mathbb{R}$.
- **Vectors**: Bold lowercase letters $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$.
- **Matrices**: Uppercase letters $A, B \in \mathbb{R}^{n \times m}$.
- **Random Variables**: Uppercase letters X, Y.
- **Probability Distributions**: $\mathbb{P}$ denotes probability measures, and $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ represents a Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ.
- **Expectation**: $\mathbb{E}[\cdot]$ denotes the expectation operator.
- **State Space**: $\mathcal{S}$ represents the set of all possible states.
- **Action Space**: $\mathcal{A}$ represents the set of all possible actions.
- **Observation Space**: $\mathcal{O}$ represents the set of all possible observations.
- **Hypotheses Space**: $\mathcal{H}$ denotes the set of all possible hypotheses regarding the environment.

- **Belief State**: b_t represents the belief state at time t, which is a probability distribution over states or hypotheses.
- **Time Index**: Discrete time steps are denoted by $t \in \mathbb{N}$.

3 Ontology $\mathcal{O}$

An **ontology** $\mathcal{O}$ is defined as a directed graph $G = (V, E)$, where:

- V is a set of nodes representing concepts, types, theorems, higher inductive types, and other mathematical entities.
- $E \subseteq V \times V$ is a set of directed edges representing relationships such as *contains*, *depends on*, *uses*, and *references*.

3.1 Structure of the Ontology

The ontology comprises the following components:

1. **Universes**: Denoted by U_i, where $i \in \mathbb{N}$, representing hierarchical levels of types. Each universe is related to the next higher universe, i.e., $U_i : U_{i+1}$.
2. **Types**: Mappings $Type : U_i \to U_{i+1}$, representing types within universes.
3. **Terms**: Elements of types, denoted as $Term(A)$ for a type A.
4. **Theorems**: Nodes representing propositions and their proofs, such as the Univalence Axiom.
5. **Higher Inductive Types (HITs)**: Types defined by constructors, e.g., the circle $\mathbb{S}^1$ with base point and loop path constructors.
6. **Meta-Mathematical Concepts**: Nodes representing meta-theorems like Gödel's Incompleteness Theorems.

3.2 Ontology Navigation Actions

The agent can perform the following actions to interact with the ontology:

- **MoveTo**$(v \in V)$: Transition to node v if $(u, v) \in E$, where u is the current node.
- **Inspect**$(v \in V)$: Retrieve attributes (definitions, theorems, descriptions) associated with node v.

3.3 Mathematical Formalization of Ontology Navigation

Let the agent's current ontology state be $n_t \in V$ at time t. The set of possible ontology actions $\mathcal{A}_{\mathcal{O}}(n_t)$ is defined as:

$$\mathcal{A}_{\mathcal{O}}(n_t) = \{\text{MoveTo}(v) \mid (n_t, v) \in E\} \cup \{\text{Inspect}(v) \mid v \in V\}. \tag{1}$$

4 Environment $\mathcal{E}$

We define a continuous navigation environment $\mathcal{E}$ as a 2D Euclidean space with obstacles and a goal position.

4.1 State Space $\mathcal{S}$

The agent's state at time t is $\mathbf{s}_t \in \mathcal{S} \subset \mathbb{R}^2$, representing its position in the 2D space.

4.2 Action Space $\mathcal{A}$

The agent's actions are acceleration vectors $\mathbf{a}_t \in \mathcal{A} \subset \mathbb{R}^2$, dictating changes in velocity.

4.3 Dynamics Model

The environment dynamics are governed by the following discrete-time kinematic equations:

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{v}_t \Delta t + \frac{1}{2}\mathbf{a}_t (\Delta t)^2 + \boldsymbol{\varepsilon}_t, \tag{2}$$

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{a}_t \Delta t + \boldsymbol{\eta}_t, \tag{3}$$

where:

- $\mathbf{v}_t$ is the agent's velocity at time t.
- $\mathbf{a}_t$ is the acceleration (action) applied at time t.
- Δt is the duration of each discrete time step.
- $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{motion}})$ represents motion noise affecting position.
- $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{velocity}})$ represents noise affecting velocity.

4.4 Observation Model

At each time step, the agent receives a noisy observation of its position:

$$\mathbf{o}_t = \mathbf{s}_t + \boldsymbol{\delta}_t, \tag{4}$$

where $\boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{obs}})$ denotes observation noise.

4.5 Goal and Obstacles

- **Goal Position**: Denoted by $\mathbf{g} \in \mathbb{R}^2$, representing the target location the agent aims to reach.
- **Obstacles**: Defined as a set $\mathcal{B} = \{(\mathbf{p}_i, r_i) \mid i \in \mathcal{I}\}$, where $\mathbf{p}_i \in \mathbb{R}^2$ is the position and $r_i > 0$ is the radius of the i-th obstacle.
- **Goal Threshold**: A scalar $\varepsilon > 0$ specifying the maximum allowable distance from the goal for successful termination.

5 Agent Definition

The agent operates within the environment $\mathcal{E}$, maintaining internal models and interacting with the ontology $\mathcal{O}$. Its primary objectives are to navigate towards the goal position while reasoning about its actions and verifying hypotheses through formal proofs.

5.1 Belief State

At any time step t, the agent maintains a belief state b_t representing its uncertainty about the environment and ontology. This belief state can be over:

1. **States**: $b_t(\mathbf{s}) = p(\mathbf{s}_t = \mathbf{s} \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})$.
2. **Hypotheses**: $b_t(H_i) = p(H_i \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})$, where $H_i \in \mathcal{H}$.

5.2 Available Actions

The agent's action space is discrete, comprising a finite set of acceleration vectors:

$$\mathcal{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_9\} \subset \mathbb{R}^2, \tag{5}$$

where each $\mathbf{a}_k$ corresponds to a specific directional acceleration, e.g., $\mathbf{a}_1 = (1, 1)^\top$ for diagonal acceleration upwards to the right.

5.3 Observation Encoding

An **Observation Encoder** is employed to map raw observations to a latent space suitable for integration with LLMs and MCTS. Formally, the encoder is a function:

$$f_{\text{enc}} : \mathcal{O} \to \mathbb{R}^d, \tag{6}$$

where d is the dimensionality of the embedding space.

5.4 Integration with Ontology

The agent's interaction with the ontology $\mathcal{O}$ allows it to leverage structured knowledge for informed decision-making. Actions such as *MoveTo* and *Inspect* facilitate navigation and information retrieval within the ontology graph.

6 Bayesian Inference and Kalman Filters

To maintain an accurate belief state, the agent employs Bayesian inference coupled with Kalman filters, enabling it to update beliefs based on observations and actions.

6.1 Bayesian Update

Given a prior belief $p(\mathbf{s}_t \mid \mathbf{o}_{1:t-1})$ and a new observation $\mathbf{o}_t$, the posterior belief is updated as follows:

$$p(\mathbf{s}_t \mid \mathbf{o}_{1:t}) = \frac{p(\mathbf{o}_t \mid \mathbf{s}_t) p(\mathbf{s}_t \mid \mathbf{o}_{1:t-1})}{p(\mathbf{o}_t \mid \mathbf{o}_{1:t-1})}, \tag{7}$$

where:

- $p(\mathbf{o}_t \mid \mathbf{s}_t)$ is the likelihood of observing $\mathbf{o}_t$ given state $\mathbf{s}_t$.
- $p(\mathbf{s}_t \mid \mathbf{o}_{1:t-1})$ is the prior belief.
- $p(\mathbf{o}_t \mid \mathbf{o}_{1:t-1})$ is the marginal likelihood of the observation.

6.2 Kalman Filter Equations

Assuming linear Gaussian models, the Kalman filter provides an efficient recursive solution for updating the belief state.

Prediction Step

$$\hat{\mathbf{s}}_{t|t-1} = \hat{\mathbf{s}}_{t-1|t-1} + \mathbf{a}_{t-1}\Delta t, \tag{8}$$

$$P_{t|t-1} = P_{t-1|t-1} + \Sigma_{\text{motion}}, \tag{9}$$

where:

- $\hat{\mathbf{s}}_{t|t-1}$ is the predicted state mean.
- $P_{t|t-1}$ is the predicted state covariance.
- Σ_{motion} is the motion noise covariance matrix.

Update Step

$$\mathbf{K}_t = P_{t|t-1}(P_{t|t-1} + \Sigma_{\text{obs}})^{-1}, \tag{10}$$

$$\hat{\mathbf{s}}_{t|t} = \hat{\mathbf{s}}_{t|t-1} + \mathbf{K}_t(\mathbf{o}_t - \hat{\mathbf{s}}_{t|t-1}), \tag{11}$$

$$P_{t|t} = (I - \mathbf{K}_t)P_{t|t-1}, \tag{12}$$

where:

- $\mathbf{K}_t$ is the Kalman gain.
- Σ_{obs} is the observation noise covariance matrix.
- I is the identity matrix.

7 Hypotheses and Prior Distribution

The agent considers a set of hypotheses $\mathcal{H} = \{H_1, H_2, \ldots, H_n\}$ about the environment's dynamics and the ontology's structure.

7.1 Prior Distribution

Initially, the agent assigns a uniform prior over the hypotheses:

$$p(H_i) = \frac{1}{n}, \quad \forall H_i \in \mathcal{H}. \tag{13}$$

7.2 Likelihood Function

Given an action $\mathbf{a}_t$ and observation $\mathbf{o}_t$, the likelihood under hypothesis H_i is defined as:

$$\mathcal{L}(\mathbf{o}_t \mid H_i) = p(\mathbf{o}_t \mid H_i, \mathbf{s}_t, \mathbf{a}_t). \tag{14}$$

7.3 Posterior Update

The posterior probability of each hypothesis is updated using Bayes' theorem:

$$p(H_i \mid \mathbf{o}_{1:t}) = \frac{\mathcal{L}(\mathbf{o}_t \mid H_i) p(H_i \mid \mathbf{o}_{1:t-1})}{\sum_{j=1}^{n} \mathcal{L}(\mathbf{o}_t \mid H_j) p(H_j \mid \mathbf{o}_{1:t-1})}. \tag{15}$$

8 Probabilistic World Model

The agent maintains a probabilistic model to predict future states and observations, facilitating informed decision-making.

8.1 Transition Model

Under hypothesis H_i, the transition model is defined as:

$$p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t, H_i) = \mathcal{N}(\mathbf{s}_{t+1}; \mathbf{s}_t + \mathbf{a}_t \Delta t, \Sigma_{\text{motion},i}), \tag{16}$$

where $\Sigma_{\text{motion},i}$ is the motion noise covariance specific to hypothesis H_i.

8.2 Observation Model

The observation model under hypothesis H_i is given by:

$$p(\mathbf{o}_t \mid \mathbf{s}_t, H_i) = \mathcal{N}(\mathbf{o}_t; \mathbf{s}_t, \Sigma_{\text{obs},i}). \tag{17}$$

8.3 Implementation

The probabilistic world model is encapsulated within a class structure, enabling modular interactions with other components of the agent.

9 Large Language Model (LLM)

The agent leverages Large Language Models (LLMs) to generate reasoning traces and Coq code for formal verification.

9.1 LLM Architecture Overview

The LLM employed is based on the Transformer architecture, characterized by multiple layers of self-attention and feedforward networks.

- **Transformer Blocks**: Consist of multi-head self-attention mechanisms followed by position-wise feedforward networks.
- **Self-Attention Mechanism**: Allows the model to weigh the importance of different input tokens relative to each other.
- **Positional Encoding**: Injects information about the position of tokens in the sequence.

9.2 Reasoning Trace Generation

The LLM generates reasoning traces that guide the agent's decision-making process. These traces are formulated as sequences of structured steps, which are subsequently parsed and executed.

9.3 Token Budget Constraint

Given computational and cost constraints, the LLM operates within a predefined token budget B, ensuring efficient utilization of resources. The relationship is defined as:

$$T_{\text{prompt}} + T_{\text{completion}} \leq B, \tag{18}$$

where T_{prompt} is the number of tokens in the input prompt, and $T_{\text{completion}}$ is the number of tokens generated by the LLM.

10 Homotopy Type Theory (HoTT) and Coq Verification

Homotopy Type Theory (HoTT) serves as the foundational framework for the agent's formal reasoning capabilities. The integration of Coq within a Docker environment facilitates computational verification of hypotheses derived from HoTT.

10.1 HoTT Concepts

- **Types as Spaces**: In HoTT, types are interpreted as topological spaces, and terms as points within these spaces.
- **Paths as Equalities**: Paths between points in a type correspond to equalities between terms.
- **Higher Inductive Types (HITs)**: Types defined not only by their points but also by paths and higher-dimensional structures.

10.2 Univalence Axiom

The Univalence Axiom is a cornerstone of HoTT, formalized as:

$$\forall A, B : U_i, \quad (A \simeq B) \Rightarrow (A =_{U_i} B), \tag{19}$$

where U_i denotes a universe, $A \simeq B$ signifies an equivalence between types A and B, and $A =_{U_i} B$ denotes their equality within the universe U_i.

10.3 Computational Verification with Coq

Coq, a formal proof management system, is employed to verify the correctness of hypotheses and proofs generated by the agent. The integration within a Docker environment ensures consistency and reproducibility.

10.3.1 Coq Integration Workflow

1. **Proof Generation**: The LLM generates Coq code corresponding to the reasoning trace.
2. **Verification**: Coq verifies the generated proofs, ensuring their correctness within the HoTT framework.
3. **Feedback Loop**: Verification results inform the agent's belief state and subsequent actions.

10.3.2 Docker Environment Setup

Docker containers encapsulate the Coq environment, providing isolated and consistent computational resources. The container setup includes:

- Installation of Coq and necessary dependencies.
- Configuration of SSH access for seamless interaction.
- Automation scripts for proof execution and result retrieval.

11 Token-Aware Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is adapted to account for token budget constraints, balancing exploration and exploitation while managing computational resources effectively.

11.1 Overview

Token-Aware MCTS integrates token consumption metrics into the traditional MCTS framework, ensuring that the agent's reasoning processes remain within budgetary limits.

11.2 Tree Structure

Each node in the MCTS tree encapsulates:

- **State**: The current belief state b_t.
- **Token Budget**: Remaining tokens B.
- **Visit Count**: Number of times the node has been visited, $N(s)$.
- **Total Value**: Accumulated reward, $W(s)$.
- **Child Nodes**: Successors representing possible actions.

11.3 Selection Policy

The selection of child nodes employs a modified Upper Confidence Bound (UCB) formula incorporating token efficiency:

$$U(s,a) = Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} - \lambda\frac{C(s,a)}{B}, \tag{20}$$

where:

- $Q(s,a) = \frac{W(s,a)}{N(s,a)}$ is the average reward for action a at state s.
- c is the exploration constant.
- λ is the token penalty weight.
- $C(s,a)$ is the cumulative token cost incurred by taking action a at state s.
- B is the total token budget.

11.4 Expansion

A node is expanded by generating all possible actions not yet explored from the current state, considering the remaining token budget.

11.5 Simulation (Rollout)

Simulations proceed by selecting actions based on a rollout policy until a terminal state or maximum depth is reached. The rollout accounts for token consumption and interactions with the ontology.

11.6 Backpropagation

After each simulation, rewards and visit counts are propagated back up the traversal path, adjusting for token efficiency and exploration bonuses.

12 Implementation Details

12.1 Observation Encoder

The observation encoder f_{enc} is implemented as a neural network mapping raw observations to a latent embedding space:

$$\mathbf{z}_t = f_{\text{enc}}(\mathbf{o}_t; \theta), \tag{21}$$

where θ denotes the network parameters.

12.1.1 Architecture

A simple feedforward neural network architecture is employed:

- Input Layer: Accepts a 2-dimensional observation vector.
- Hidden Layers: Two fully connected layers with ReLU activations.
- Output Layer: Outputs a 32-dimensional embedding vector.

12.2 Reward Function

The agent's reward function R_t is designed to balance environmental rewards, token penalties, exploration incentives, and reasoning quality:

$$R_t = w_{\text{env}} r_t - w_{\text{token}} \exp\left(\frac{T_t}{B}\right) + w_{\text{explore}} e_t + w_{\text{quality}} q_t, \tag{22}$$

where:

- r_t is the base environmental reward at time t.
- T_t is the cumulative tokens used up to time t.
- B is the total token budget.
- e_t is the exploration bonus.
- q_t is the reasoning quality metric.
- w_{env}, w_{token}, w_{explore}, w_{quality} are weighting factors.

12.3 LLM-TAC Wrapper

The LLM Tactical (TAC) Wrapper manages interactions with the LLM, handling prompt generation, response caching, and rate limiting.

12.3.1 Functionalities

- **Prompt Creation**: Constructs prompts based on the agent's current state and history.
- **Response Caching**: Stores previous LLM responses to minimize redundant computations and conserve token usage.
- **Rate Limiting**: Implements mechanisms to comply with API rate limits, preventing excessive token consumption.

12.4 Docker Environment

Docker containers encapsulate the execution environments for Coq and other dependencies, ensuring consistency and reproducibility across different deployment platforms.

12.4.1 Containerization

- **Coq Setup**: The Docker image includes the Coq proof assistant and necessary libraries.
- **SSH Configuration**: Secure Shell (SSH) access is configured to facilitate remote execution of Coq scripts.
- **Automation Scripts**: Scripts are provided to automate the execution and verification of Coq proofs generated by the LLM.

12.4.2 Integration Workflow

1. The LLM generates Coq code based on the reasoning trace.
2. The Coq code is uploaded to the Docker container via SSH.
3. Coq executes the proof, and the results are retrieved.
4. Verification outcomes inform the agent's belief state and future actions.

13 Agent Workflow

The AI agent operates through the following sequential steps:

1. **Observation Acquisition**: The agent receives a noisy observation $\mathbf{o}_t$ from the environment.
2. **Belief State Update**: Utilizing Bayesian inference and Kalman filters, the agent updates its belief state b_t.
3. **Reasoning Trace Generation**: The LLM generates a reasoning trace within the remaining token budget.
4. **Coq Verification**: Generated Coq code is executed within the Dockerized environment to verify hypotheses.
5. **MCTS Action Selection**: Token-Aware MCTS selects the next action $\mathbf{a}_t$ based on the current belief state and token constraints.
6. **Action Execution**: The agent executes the selected action in the environment.
7. **Reward Reception**: The agent receives a reward R_t based on the action's outcome.

14 Algorithms

14.1 Token-Aware Monte Carlo Tree Search (MCTS)

Algorithm 1 Token-Aware MCTS Selection Policy

Require: Root node R, exploration constant c, token penalty weight λ, total token budget B
Ensure: Selected action a

Initialize *current* $\leftarrow R$
while node *current* is fully expanded and not terminal **do**
  Select child with highest UCB score:

$$U(s,a) = Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} - \lambda\frac{C(s,a)}{B}$$

  current $\leftarrow$ selected child
end while
if node *current* is not fully expanded **then**
  Expand node *current* by adding a new child node with action a
  current $\leftarrow$ new child node
end if
Simulate a rollout from node *current* to obtain reward r
Backpropagate the reward r up the tree
return action associated with the best child of R

14.2 Kalman Filter Update

Algorithm 2 Kalman Filter Update Step

Require: Prior mean $\hat{\mathbf{s}}_{t|t-1}$, prior covariance $P_{t|t-1}$, observation $\mathbf{o}_t$, observation noise covariance Σ_{obs}
Ensure: Updated mean $\hat{\mathbf{s}}_{t|t}$, updated covariance $P_{t|t}$

Compute Kalman Gain:

$$\mathbf{K}_t = P_{t|t-1}(P_{t|t-1} + \Sigma_{\text{obs}})^{-1}$$

Update mean:

$$\hat{\mathbf{s}}_{t|t} = \hat{\mathbf{s}}_{t|t-1} + \mathbf{K}_t(\mathbf{o}_t - \hat{\mathbf{s}}_{t|t-1})$$

Update covariance:

$$P_{t|t} = (I - \mathbf{K}_t)P_{t|t-1}$$

return $\hat{\mathbf{s}}_{t|t}$, $P_{t|t}$

14.3 Bayesian Posterior Update

Algorithm 3 Bayesian Posterior Update

Require: Current hypotheses $\mathcal{H}$, prior probabilities $p(H_i)$, action $\mathbf{a}_t$, observation $\mathbf{o}_t$
Ensure: Updated posterior probabilities $p(H_i \mid \mathbf{o}_{1:t})$
1: **for** each hypothesis $H_i \in \mathcal{H}$ **do**
2: Predict next state $\hat{\mathbf{s}}_{t|t-1} = \mathbf{s}_t + \mathbf{a}_t \Delta t$
3: Compute likelihood $\mathcal{L}(\mathbf{o}_t \mid H_i) = \mathcal{N}(\mathbf{o}_t; \hat{\mathbf{s}}_{t|t-1}, \Sigma_{\text{obs},i})$
4: **end for**
5: Compute evidence $p(\mathbf{o}_t \mid \mathbf{o}_{1:t-1}) = \sum_i \mathcal{L}(\mathbf{o}_t \mid H_i) p(H_i)$
6: **for** each hypothesis $H_i \in \mathcal{H}$ **do**
7: Update posterior:

$$p(H_i \mid \mathbf{o}_{1:t}) = \frac{\mathcal{L}(\mathbf{o}_t \mid H_i) p(H_i)}{p(\mathbf{o}_t \mid \mathbf{o}_{1:t-1})}$$

8: **end for**
9: **return** $p(H_i \mid \mathbf{o}_{1:t})$ for all H_i

15 Formalization and Theoretical Framework

15.1 Agent's Decision-Making Process

The agent's decision-making process integrates probabilistic inference, structured knowledge representation, and formal verification. The interaction among these components can be formalized as follows:

1. **Perception**: At each time step t, the agent receives an observation $\mathbf{o}_t$.
2. **Belief Update**: The agent updates its belief state b_t using Bayesian inference and Kalman filtering.
3. **Reasoning Generation**: The LLM generates a reasoning trace τ_t based on the current belief and ontology state.
4. **Formal Verification**: The reasoning trace is translated into Coq code and verified within the Dockerized environment.
5. **Action Selection**: Token-Aware MCTS selects the optimal action $\mathbf{a}_t$ within the token budget.
6. **Execution**: The agent executes the selected action, transitioning to a new state $\mathbf{s}_{t+1}$.
7. **Reward Reception**: The agent receives a reward R_t based on the action's outcome.

15.2 Mathematical Models

15.2.1 Ontology Representation

The ontology $\mathcal{O}$ is a directed graph $G = (V, E)$, with nodes representing various mathematical constructs and edges denoting relationships. Formally, it can be represented as:

$$G = (V, E), \quad V = \{v_1, v_2, \ldots, v_n\}, \quad E \subseteq V \times V$$

15.2.2 Probabilistic Models

State Transition Under hypothesis H_i, the state transition is modeled as:

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) + \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{motion},i})$$

Observation Model Observations are modeled as:

$$\mathbf{o}_t = h(\mathbf{s}_t) + \boldsymbol{\delta}_t, \quad \boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{obs},i})$$

15.2.3 Bayesian Inference

The posterior probability of a hypothesis H_i given observations $\mathbf{o}_{1:t}$ and actions $\mathbf{a}_{1:t-1}$ is:

$$p(H_i \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1}) = \frac{p(\mathbf{o}_t \mid H_i, \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}) p(H_i \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1})}{p(\mathbf{o}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1})}$$

15.3 Integration with Homotopy Type Theory

HoTT provides a robust framework for reasoning about types and their equivalences. By formalizing the agent's reasoning processes within HoTT and verifying them using Coq, we ensure the correctness and reliability of the agent's actions.

[Univalence Axiom] For any two types $A, B \in U_i$, if there exists an equivalence $A \simeq B$, then A and B are equal in U_i, i.e.,

$$(A \simeq B) \Rightarrow (A =_{U_i} B)$$

Proof omitted for brevity. Refer to HoTT literature for detailed proof. □

16 Coq Integration

In this section, we detail the integration of the Coq proof assistant within our AI agent's framework. Coq is used for the computational verification of hypotheses derived from Homotopy Type Theory (HoTT). The integration ensures that the agent's reasoning and decision-making processes are grounded in formally verified mathematics.

16.1 Coq as a Formal Verification Tool

Coq is an interactive theorem prover that allows for the expression of mathematical assertions, the mechanical verification of proofs, and the extraction of certified programs. By leveraging Coq, the agent can verify the correctness of its reasoning steps, particularly those involving complex mathematical constructs from HoTT.

16.2 Dockerized Coq Environment

To facilitate seamless and reproducible integration, Coq is run within a Docker container. This ensures a consistent environment across different systems and simplifies deployment.

16.2.1 Docker Setup

The Docker environment includes:

- **Base Image**: An official Coq image or a minimal Linux distribution with Coq installed.
- **Coq Installation**: The latest version of Coq compatible with the agent's requirements.
- **SSH Configuration**: Enables remote execution of Coq scripts from the agent.
- **Dependency Management**: All necessary libraries and dependencies, including those for HoTT.

16.2.2 Container Configuration

The container exposes necessary ports for SSH communication and ensures security measures are in place to prevent unauthorized access.

16.3 Workflow for Coq Verification

The agent's interaction with Coq follows a structured workflow:

1. **Generation of Coq Code**: The agent, via the LLM, generates Coq scripts corresponding to the reasoning steps or hypotheses it needs to verify.
2. **Uploading to Docker Container**: The generated Coq scripts are securely transferred to the Docker container using SSH.
3. **Execution of Proofs**: Within the container, Coq executes the scripts, attempting to verify the proofs.
4. **Result Retrieval**: The outcome of the proof (success or failure, along with any error messages) is retrieved by the agent.
5. **Updating Belief State**: The agent updates its belief state based on the verification results. Successful proofs reinforce the agent's confidence in certain hypotheses, while failures may prompt revisions in reasoning.

16.4 Example Integration

Suppose the agent needs to verify a property about a higher inductive type in HoTT, such as the homotopy group of the circle $\mathbb{S}^1$.

1. **LLM Generates Coq Code**:

Listing 1: Coq Code for Verifying $\pi_1(\mathbb{S}^1) \cong \mathbb{Z}$

```
Require Import HoTT.

Theorem pi1_S1 : forall (x : Sphere1), GroupIsomorphism (pi1 x) Z.
Proof.
  (* Proof steps here *)
Admitted.
```

2. **Code Upload**: The script is uploaded to the Coq Docker container.
3. **Proof Execution**: Coq attempts to complete the proof.
4. **Result Retrieval**: If the proof is incomplete (as indicated by `Admitted`), the agent recognizes that further reasoning is needed.
5. **Belief Update**: The agent adjusts its hypotheses or seeks additional information from the ontology.

16.5 Handling Verification Failures

When a proof attempt fails, the agent employs strategies such as:

- **Refinement of Reasoning**: Analyzing error messages to identify gaps in the proof and refining the reasoning steps accordingly.
- **Consulting the Ontology**: Navigating the ontology to find relevant theorems, definitions, or lemmas that may aid in completing the proof.
- **Adjusting Hypotheses**: Revising or selecting alternative hypotheses that are more consistent with the verified knowledge.

16.6 Security and Resource Management

The integration considers security and resource constraints:

- **Sandboxing**: The Docker container isolates the Coq environment, preventing interference with the host system.
- **Resource Limits**: The container is configured with limits on CPU and memory usage to prevent overconsumption of resources.
- **Authentication**: SSH keys and secure passwords are used to authenticate communication between the agent and the container.

17 Agent Workflow and System Architecture

The AI agent operates through a sequence of steps that integrate all the components discussed. The system architecture ensures that each component interacts cohesively to achieve the agent's objectives.

17.1 Agent Workflow

1. **Perception**: The agent receives observations from the environment and encodes them using the observation encoder.
2. **Belief Update**: Bayesian inference and Kalman filters are used to update the agent's belief state based on new observations.
3. **Reasoning**: The LLM generates reasoning traces and potential actions, considering the agent's goals and the current belief state.
4. **Verification**: Generated reasoning is translated into Coq code and verified within the Dockerized Coq environment.
5. **Planning**: Token-Aware MCTS uses the verified reasoning to plan actions, taking into account the token budget.
6. **Action Execution**: The selected action is executed in the environment.
7. **Ontology Interaction**: The agent navigates the ontology to gather more information or update its knowledge base.
8. **Iteration**: The cycle repeats, allowing the agent to continuously adapt and refine its strategies.

17.2 System Architecture

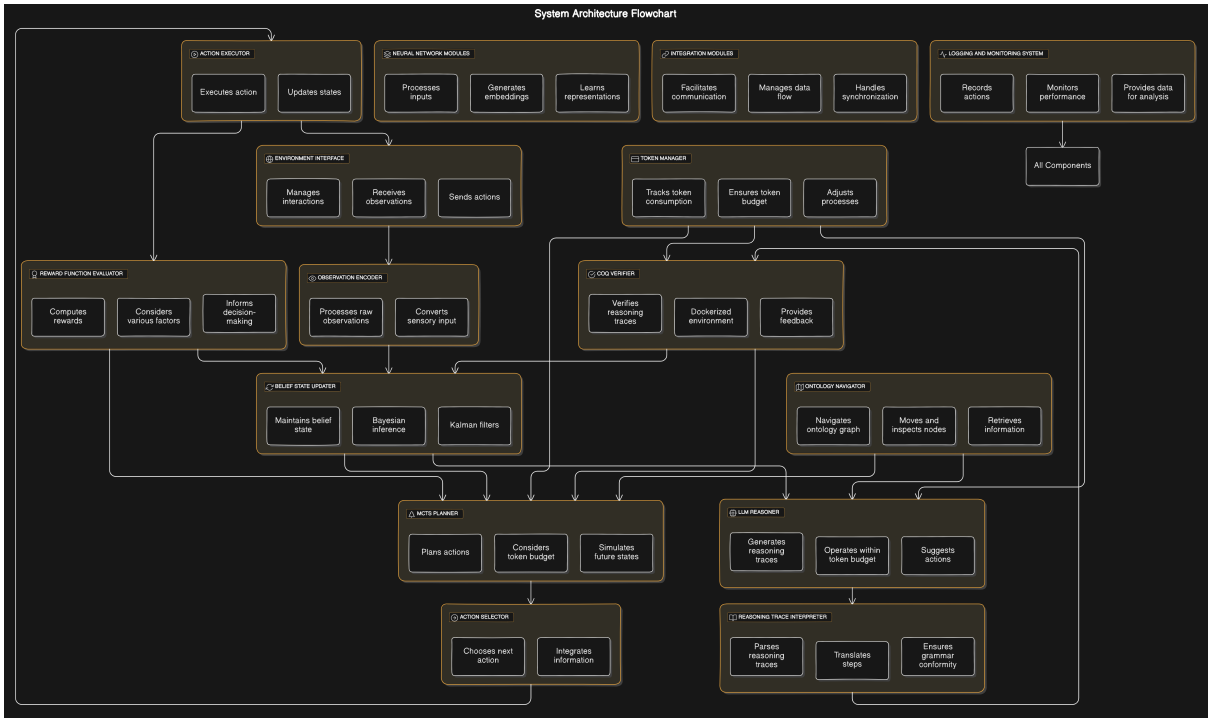

Figure 1: System Architecture of the AI Agent

17.2.1 Components

- **Environment**: The external world where the agent operates, providing observations and receiving actions.
- **Observation Encoder**: Transforms raw observations into encoded representations suitable for the agent's internal processing.
- **Bayesian Inference Module**: Updates the belief state using probabilistic models.
- **LLM Reasoning Module**: Generates reasoning traces and suggests actions based on the current belief state and goals.
- **Coq Verification Module**: Verifies the reasoning using formal methods.
- **Token-Aware MCTS Planner**: Plans actions while considering the token budget.
- **Ontology Navigator**: Interacts with the ontology for knowledge retrieval and updates.
- **Action Executor**: Executes selected actions in the environment.

17.2.2 Data Flow

1. Observations flow from the environment to the observation encoder.
2. Encoded observations are used by the Bayesian inference module to update beliefs.
3. The updated belief state informs the LLM reasoning module.
4. Reasoning outputs are sent to the Coq verification module.
5. Verified reasoning informs the MCTS planner.
6. The planner selects actions, which are executed by the action executor.
7. The ontology navigator provides additional knowledge to both the LLM and the planner.

18 Conclusion

We have presented a formal mathematical framework for an AI agent that integrates ontologies, Bayesian inference, Kalman filters, large language models, token-aware Monte Carlo Tree Search, and Homotopy Type Theory. By incorporating the Coq proof assistant within a Dockerized environment, the agent is capable of computationally verifying its reasoning steps, ensuring that its decision-making processes are both robust and mathematically sound.

The agent operates in a continuous navigation environment, utilizing advanced probabilistic models to update its belief state and make informed decisions under uncertainty. The integration with an ontology allows the agent to leverage structured knowledge, while the use of HoTT and formal verification through Coq provides a strong foundation for reasoning about complex mathematical concepts.

References

[1] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.

[2] Bertot, Y., and Castéran, P. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer Science & Business Media, 2013.

[3] Browne, C. B., et al. "A Survey of Monte Carlo Tree Search Methods." *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2012, pp. 1–43.

[4] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems." *Journal of Basic Engineering*, vol. 82, no. 1, 1960, pp. 35–45.

[5] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[6] Radford, A., et al. "Language Models are Unsupervised Multitask Learners." OpenAI Technical Report, 2019.

[7] Merkel, D. "Docker: Lightweight Linux Containers for Consistent Development and Deployment." *Linux Journal*, vol. 2014, no. 239, 2014.

[8] Coq Docker Images. Available at: `https://github.com/coq-community/docker-coq`