A MĪMĀṀSĀ INSPIRED FRAMEWORK FOR INSTRUCTION SEQUENCING IN AI AGENTS

Abstract.

1. Introduction

Assume a robot is given the task of *"cooking tomato noodles stir-fry"*. Though it may appear simple at first glance, there are several intricate details that must be considered before executing each instruction. For instance, this task may consist of several steps such as *"pick noodles, cook noodles in pot, chop tomato, fry tomato, and add the cooked noodle to a dish"*. The robot must determine what actions to perform, which objects to use, and the appropriate order in which to execute them. To logically determine the sequence with the integration of objects is a challenge.

This paper addresses this challenge through novel framework for instruction sequencing inspired by principles from the Indian philosophical system of Mīmāṁsā[1]. In this approach, each instruction is represented as an $< \text{action}, \text{object} >$ pair, which serves as the foundation for evaluating consistency across subsequent instructions. An initial version of this work was presented and published in the proceedings of ICLA 2025, where the fundamental representation of instructions as action–object pairs was introduced [**?**]. The present paper significantly extends that work by formalizing the criteria for valid instruction sequencing, including the introduction of an object consistency theorem with soundness and completeness.

The remainder of the paper is structured as follows: Section **??** outlines the logical formalism of MIRA with syntax and semantics, which forms the basis for the sequencing methods. Section **??** presents the representation of instructions as `action, object` pairs with sequencing mechanisms. The various sequencing strategies—including Direct Assertion, Purpose-Based Sequencing, and the Sequential Completion / Iterative Procedure—are formally described in Section 4. The validity of these sequencing methods is established in Section 5. Related work is reviewed in Section **??**, and the specific advancements made over prior approaches are summarized in Section **??**. Finally, Section **??** presents the conclusion.

[1]Mīmāṁsā is a classical Indian philosophical system that developed a detailed theory of imperatives (vidhi), focusing on their function, classification, and execution of actions in a systematic procedure. Its procedural system has recently attracted attention in computational contexts due to its fine-grained treatment of instruction structure and intent.

Preprint submitted to
Logical Methods in Computer Science

2. Extension of Syntax and Semantics for Action-Object Imperative Logic

This section extends the formal syntax and semantics framework of Srinivasan and Parthasarathi (2021) [**?**] by refining imperative instructions to explicit action-object pairs.

2.1. **Syntax.**

The language of imperatives is given by $\mathcal{L}_i = \langle I, R, P, B \rangle$, where:

- $I = \{i_1, i_2, \ldots, i_n\}$ is the set of imperatives,
- $R = \{r_1, r_2, \ldots, r_n\}$ is the set of reasons,
- $P = \{p_1, p_2, \ldots, p_n\}$ is the set of purposes (goals),
- $B = \{\wedge, \vee, \rightarrow_r, \rightarrow_i, \rightarrow_p\}$ is the set of binary connectives.

Here, R and P are propositions, following propositional formula syntax. They combine with imperatives I in several forms to build Imperative Formulas $\mathcal{F}_i$, specified by Equation 2.3:

$$\mathcal{F}_i = \{i \mid i \rightarrow_p p \mid (i \rightarrow_p p_1) \wedge (j \rightarrow_p p_2) \mid (i \rightarrow_p \theta) \oplus (j \rightarrow_p \theta) \mid (\varphi \rightarrow_i \psi) \mid (\tau \rightarrow_r \varphi)\} \quad (2.1)$$

The fundamental unit of an imperative, denoted i, decomposes into an action and a set of objects. For example, the instruction *"Take a book"* associates the action *"take"* with the object *"book"*.

Formally, this association is a function:

$$f : I \rightarrow A \times \mathcal{P}(O),$$

where $I = \{i_1, i_2, i_3, \ldots, i_n\}$ is a set of instructions, $A = \{a_1, a_2, \ldots, a_k\}$ is a set of actions, and $O = \{o_1, o_2, \ldots, o_m\}$ is a set of objects.

Each instruction $i_j \in I$ can be represented as:

$$i_j = (a_j, o_j) \quad (2.2)$$

where $a_j \in A$, and $o_j \subseteq O$.

Here, each action can stand alone, or be paired with zero, one, or multiple objects as summarized in Table 2.1.

Action Type	**Representation**	**Example**
Action with object	$(a_j, \{o_k\})$	$(\texttt{pick}, \{\texttt{rice}\})$ - *pick rice*
Action with multiple objects	$(a_j, \{o_k, o_m\})$	$(\texttt{cook}, \{\texttt{rice}, \texttt{pot}\})$ - *cook rice in pot*
Action without object	$(a_j, \emptyset)$	$(\texttt{wait}, \emptyset)$ - *wait*

Table 1: Instruction representations for actions paired with zero, one, or multiple objects.

2.2. **Semantic Model.**

A semantic model is defined as $\mathcal{M}$ as in Equation 2.6.

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{G}, intention, eval \rangle \quad (2.3)$$

where, $\mathcal{S}$: set of system states, $\mathcal{A}$: set of actions, $\mathcal{O}$: set of objects, *intention*: $\mathcal{S}X\mathcal{G} \rightarrow \{true, false\}, \mathcal{S}X(\mathcal{A}X\mathcal{O}) \rightarrow \{S, V, N\}$. The semantic evaluation for different imperatives are given as follows:

- Unconditional Imperatives:

$$\text{eval}(s, (a, o)) = \begin{cases} S & \text{if the action } a \text{ is successfully performed on } o \text{ in } s, \\ V & \text{otherwise.} \end{cases}$$

- Imperative enjoining goal:

$$\text{eval}(s,(a,o)\rightarrow_p g)=\begin{cases}S & \text{if the agent intends } g \text{ and } (a,o) \text{ is satisfied in } s\\ V & \text{if the agent intends} v \text{ and } (a,o) \text{ is violated}\\ N & \text{if there is no intention to achieve the goal } g\end{cases}$$

- Imperatives in sequence:

$$\text{eval}(s,(a_1,o_1)\rightarrow_i (a_2,o_2))=\begin{cases}S, \text{ if eval } (s,(a_1,o_1)=S \text{ and}\\ \text{eval}(s',(a_2,o_2))=S \text{ and}\\ \text{ object consistency holds as per Equation } \textbf{??}\\ V, \text{ if either eval } (s,(a_1,o_1)=V) \text{ or}\\ \text{ eval } (s,(a_2,o_2)=V\end{cases}$$

- Imperatives in parallel

$$\text{eval}((s_1,(a_1,o_1))\wedge(s_2,(a_2,o_2)))=\begin{cases}S, \text{ if eval } (s_1,(a_1,o_1)=S \text{ and}\\ \text{eval}(s_2,(a_2,o_2))=S \text{ and}\\ \text{ object consistency does not hold as per Equation } \textbf{??}\\ V, \text{ if either eval } (s,(a_1,o_1)=V) \text{ or}\\ \text{ eval } (s,(a_2,o_2)=V\end{cases}$$

- Imperatives with choice

$$\text{eval}((s_1,(a_1,o_1))\oplus(s_2,(a_2,o_2)))=\begin{cases}S, & \text{if eval}(s_1,(a_1,o_1))=S \text{ or eval}(s_2,(a_2,o_2))=V\\ S, & \text{if eval}(s_1,(a_1,o_1))=V \text{ or eval}(s_2,(a_2,o_2))=S\\ V, & \text{if eval}(s_1,(a_1,o_1))=S \text{ and eval}(s_2,(a_2,o_2))=S\\ V, & \text{if eval}(s_1,(a_1,o_1))=V \text{ and eval}(s_2,(a_2,o_2))=V\end{cases}$$

The syntax of the imperative language specifies the formation rules for well-formed imperative formulas by combining imperaties, binary connectives, reasons, and goals.

2.3. **Syntax.**

2.4. **Semantics.** The semantics defines how each imperative formula is interpreted over the model, assigning it a satisfaction status based on system states, actions, objects, and goal-directed intentions. A semantic model $\mathcal{M}$ is defined as:

$$\mathcal{M}=\langle \mathcal{S},\mathcal{A},\mathcal{O},\mathcal{G},intention,eval\rangle, \tag{2.4}$$

where

- $\mathcal{S}$ is the set of system states,
- $\mathcal{A}$ is the set of actions,
- $\mathcal{O}$ is the set of objects,
- $\mathcal{G}$ is the set of goals,
- $intention : \mathcal{S}\times\mathcal{G}\rightarrow\{true,false\}$,
- $eval : \mathcal{S}\times(\mathcal{A}\times\mathcal{O})\rightarrow\{S,V,N\}$.

The semantic evaluations for different imperatives are given as follows:

- **Unconditional Imperatives**:

$$eval(s,(a,o)) = \begin{cases} S, & \text{if the action } a \text{ is successfully performed on } o \text{ in } s, \\ V, & \text{otherwise.} \end{cases}$$

- **Imperative Enjoining Goal**:

$$eval(s,(a,o) \rightarrow_p g) = \begin{cases} S, & \text{if the agent intends } g \text{ and } (a,o) \text{ is satisfied in } s, \\ V, & \text{if the agent intends } g \text{ and } (a,o) \text{ is violated in } s, \\ N, & \text{if there is no intention to achieve goal } g. \end{cases}$$

- **Imperatives in Sequence**:

$$eval(s,(a_1,o_1) \rightarrow_i (a_2,o_2)) = \begin{cases} S, & \text{if } eval(s,(a_1,o_1)) = S,\ eval(s',(a_2,o_2)) = S \text{ and object consistency holds,} \\ V, & \text{if } eval(s,(a_1,o_1)) = V \text{ or } eval(s',(a_2,o_2)) = V. \end{cases}$$

- **Imperatives in Parallel**:

$$eval((s_1,(a_1,o_1)) \wedge (s_2,(a_2,o_2))) = \begin{cases} S, & \text{if } eval(s_1,(a_1,o_1)) = S, \quad eval(s_2,(a_2,o_2)) = S, \\ & \text{and object consistency does not hold,} \\ V, & \text{if } eval(s_1,(a_1,o_1)) = V \text{ or } eval(s_2,(a_2,o_2)) = V. \end{cases}$$

- **Imperatives with Choice**:

$$eval((s_1,(a_1,o_1)) \oplus (s_2,(a_2,o_2))) = \begin{cases} S, & \text{if } eval(s_1,(a_1,o_1)) = S \text{ or } eval(s_2,(a_2,o_2)) = S, \\ V, & \text{if } eval(s_1,(a_1,o_1)) = V \text{ and } eval(s_2,(a_2,o_2)) = V. \end{cases}$$

Equation 2.3 encompasses various types of imperatives [2] commonly found in real-world scenarios.

3. Action-Object Mapping

This mapping of instruction to action and objects help in sequencing, as detailed in the next section.

4. Sequencing Methods

According to the Indian philosophical system of Mīmāṁsā, a set of instructions can be sequenced using six distinct ordering principles to ensure coherent and uninterrupted execution. These are: Direct Assertion (*Śrutikrama*), Purpose-Based Sequencing (*Arthakrama*), Order as Given (*Pāṭhakrama*), Position-Based Order (*Sthānakrama*), Principal Activity-Based Order (*Mukhyakrama*), and Iterative Procedure (*Pravṛttikrama*). For more details on the sequencing aspects from the philosophy, the reader may refer to the prior work on temporal ordering of instructions [**?**]. Among these, three methods—Direct Assertion, Purpose-Based Sequencing, and Iterative Procedure—are formalized in this paper and discussed in Sections 4.1, 4.2, and **??**, respectively.

[2]The terms 'imperatives' and 'instructions' are used interchangeably in this paper.

4.1. **Direct Assertion (Śrutikrama).** In this type, instructions are provided in a direct and sequential manner. Following the notation from the work of sequencing methods based on M̄imāṁsā [**?**], let:

- $i_t = (a_t, o_t)$: instruction at time t, with action a_t and object(s) o_t
- $i_{t+1} = (a_{t+1}, o_{t+1})$: instruction at time t+1

Then, Instruction in sequence can be expressed by Equation 4.1.

$$(a_t o_t) \rightarrow_i (a_{t+1} o_{t+1}) \tag{4.1}$$

where $\rightarrow_i$ denotes temporal sequencing of actions on objects. The indication can be read as "perform a_t on o_t, then perform a_{t+1} on o_{t+1}".

For a sequence of n instructions, Equation 4.1 can be extended as shown in Equation 4.2.

$$(a_1 o_1) \rightarrow_i (a_2 o_2) \rightarrow_i ... \rightarrow_i (a_n o_n) \tag{4.2}$$

This representation indicates that each instruction must be completed before the next instruction.

For example, consider three statements *"pick rice"*, *"cook rice in pot"*, *"add rice to dish"*. These can be represented as:

$$(pick\{rice\}) \rightarrow_i (cook\{rice, pot\}) \rightarrow_i (add\{rice, dish\}) \tag{4.3}$$

Here, objects are progressively updated across instructions. For instance, "rice" becomes "cooked rice" after executing the instruction "cook rice." This transformation indicates that the instructions are linked through evolving object states, a relationship known as **object dependency**.

This type of representation serves two major purposes.

(1) It indicates the temporal order of the actions.
(2) The dependencies of objects the across each step is enforced.

4.2. **Sequencing based on purpose.** In this type, each instruction is of the form $(\tau \rightarrow_r (i \rightarrow_p) p)$ [**?**], indicating there is a ground or reason (τ) for the instruction (i) to take place, inorder to achieve the goal (p). Here, $\rightarrow_r$ and $\rightarrow_p$ denote "because of reason" and "inorder to achieve goal", respectively.

This representation can be extended to a series of instructions as given by Equation 4.4.

$$(r_1 \rightarrow_r (i_1 \rightarrow_p p_1)), (r_2 \rightarrow_r (i_2 \rightarrow_p p_2), ..., (r_n \rightarrow_r (i_n \rightarrow_p p_n)) \tag{4.4}$$

If the purpose p_k of instruction i_k becomes the reason r_{k+1} for the next instruction i_{k+1}, then i_k precedes i_{k+1}, because $r_{k+1} = p_k$. This relation signifies that the second insruction (i_{k+1}) depends on the first (i_k) and is referred to as **functional dependency** and has already been used in task analysis for special education [**?**].

Extending this further into the representation of i_j as (a_j, o_j) pair, Equation 4.4 can be formalized as shown in Equation 4.5.

$$\forall j \in \{1, ..., n-1\} : r_j \rightarrow_r ((a_j, o_j) \rightarrow_p p_j), r_{j+1} = p_j \tag{4.5}$$

This representation creates object dependency in addition to functional dependency, thereby strengthening the sequencing method.

4.2.1. *Sequential Completion Method.* In this method, the full sequence is performed on one object and the same sequence is repeated for all other objects.

Formally, it can be represented as follows:

Let there be n actions $A = \{a_1, a_2, ..., a_n\}$ and T objects for each action, $O_k = \{o_{k1}, o_{k2}, ..., o_{kT}\}$ for $1 \leq k \leq n$.

For each object $o_j (1 \leq j \leq T)$, the sequence is given by Equation 4.6.

$$(a_1 o_{1j} \rightarrow_i a_2 o_{2j} \rightarrow_i ... \rightarrow_i a_n o_{nj}) \tag{4.6}$$

This is repeated for all j as shown below.

$$\begin{aligned} &(a_1 o_{11} \rightarrow_i a_2 o_{21} \rightarrow a_3 o_{31} \rightarrow_i ... \rightarrow_i a_n o_{n1}) \\ &(a_1 o_{12} \rightarrow_i a_2 o_{22} \rightarrow a_3 o_{32} \rightarrow_i ... \rightarrow_i a_n o_{n2}) \\ &(a_1 o_{13} \rightarrow_i a_2 o_{23} \rightarrow a_3 o_{33} \rightarrow_i ... \rightarrow_i a_n o_{n3}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ &(a_1 o_{1T} \rightarrow_i a_2 o_{2T} \rightarrow a_3 o_{3T} \rightarrow_i ... \rightarrow_i a_n o_{nT}) \end{aligned} \tag{4.7}$$

Equation 4.7 can be interpreted as:

- For each object j, all actions are performed in sequence before moving to the next object.
- The objects involved in sequence are $(o_{1j}, o_{2j}, ...o_{nj})$ for j.

4.2.2. *Step-by-Step Parallel Method (Iterative Procedure).* In this method, the first action is performed on all objects, followed by second action and so on. Same action is grouped and distributed across objects before moving to the next action.

Formally, each action a_k $(1 \leq k \leq n)$ performed across T objects $(1 \leq j \leq T)$ is represented as:

$$(a_k o_{k1} \rightarrow_i a_k o_{k2} \rightarrow_i, ..., \rightarrow_i a_k o_{kT}) \tag{4.8}$$

This Equation when extended to all actions is represented as:

$$\begin{aligned} &(a_1 o_{11} \rightarrow_i a_1 o_{12} \rightarrow a_1 o_{13} \rightarrow_i ... \rightarrow_i a_1 o_{1T}) \\ &(a_2 o_{21} \rightarrow_i a_2 o_{22} \rightarrow a_2 o_{23} \rightarrow_i ... \rightarrow_i a_2 o_{2T}) \\ &(a_3 o_{31} \rightarrow_i a_3 o_{32} \rightarrow a_3 o_{33} \rightarrow_i ... \rightarrow_i a_3 o_{3T}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ &(a_n o_{n1} \rightarrow_i a_n o_{n2} \rightarrow a_n o_{n3} \rightarrow_i ... \rightarrow_i a_n o_{nT}) \end{aligned} \tag{4.9}$$

Equation 4.9 can be interpreted as follows:

- For each action k, all objects $o_1, o_2, ..., o_T$ are processed before moving to the next action.
- Objects involved in each action group are $O_k = \{o_{k1}, o_{k2}, ..., o_{kT}\}$

Using these sequencing mechanisms, validity can be logically determined through object dependency and consistency across subsequent instructions, as detailed in the following section.

5. Validity of Instruction Sequencing

A set of instructions is considered to be in a valid sequence if it satisfies the conditions of object dependency and state-wise consistency of subsequent instructions. These conditions are defined below.

Definition 1. ***Dependency Condition***
For a valid dependency between i_j and i_{j+1}, $\exists o^* \in O_j \cap O_{j+1}$.
That is, there exists at least one object o^* that is present in both O_j and O_{j+1}.

Definition 2. ***Consistency Condition for Sequential Instructions***
Let

- *$i_j = (a_j, O_j)$ be the j-th instruction, where a_j is the action and $O_j \subseteq O$ is the set of objects involved.*
- *$i_{j+1} = (a_{j+1}, O_{j+1})$ be the next instruction in the sequence.*
- *For each object o^*, let $s_j(o^*)$ denote the state of o^* immediately after executing i_j, and $s_{j+1}^{\text{req}}(o^*)$ denote the required state of o^* for executing i_{j+1}.*

*Then the **consistency condition** holds between i_j and i_{j+1} if:*

$$\exists\, o^* \in O_j \cap O_{j+1} \;\; \textit{such that} \;\; s_j(o^*) = s_{j+1}^{\text{req}}(o^*)$$

This means that there exists at least one object o^* that is present in both instructions, and the state of o^* after executing i_j matches the required state for i_{j+1}.

With these conditions, the theorem for Valid Instruction Sequencing is formalized.

Theorem 1. ***Object Consistency Theorem for Valid Instruction Sequencing***
Given a sequence of instructions $I = \{i_1, i_2, \ldots, i_n\}$, where each $i_j = (a_j, O_j)$, for every pair of consecutive instructions (i_j, i_{j+1}) where a dependency exists, $\exists o^* \in O_j \cap O_{j+1}$ and the state of o^* after i_j matches the required state for i_{j+1}, then the sequence is valid with respect to object dependencies.

Proof. The proof of the theorem follows by induction on the consistency maintained between instructions.

Base Case:

For the first instruction $i_1 = (a_1, O_1)$:

- Each object $o \in O_1$ is in its initial state, denoted $s_1^{\text{init}}(o)$.
- Since there is no prior instruction, there are no dependencies to check.
- The instruction i_1 can be executed as long as its required preconditions (the states of objects in O_1) are satisfied by their initial states.

Inductive Step:

Assume that for all instructions up to step j, the following holds:

- For every pair (i_k, i_{k+1}) with $1 \leq k \leq j-1$, if a dependency exists (i.e., there is at least one shared object $o^* \in O_k \cap O_{k+1}$), then the consistency condition is satisfied:

$$s_k(o^*) = s_{k+1}^{\text{req}}(o^*)$$

That is, the state of o^* after executing i_k matches the required state for i_{k+1}.

Now, consider the next instruction $i_{j+1} = (a_{j+1}, O_{j+1})$:

- **Dependency Check:** If there exists at least one object $o^* \in O_j \cap O_{j+1}$, then i_{j+1} depends on i_j for that object.

- **Consistency Condition:** The state of o^* after executing i_j, denoted $s_j(o^*)$, must equal the required state for i_{j+1}, denoted $s^{\text{req}}_{j+1}(o^*)$:

$$s_j(o^*) = s^{\text{req}}_{j+1}(o^*)$$

- **Conclusion:** If this condition holds for all such dependencies, then i_{j+1} can be executed validly, and the sequence up to i_{j+1} maintains object state consistency.

□

This theorem can be formalized as follows:

$$\forall j \in \{1, \ldots, n-1\},\ \text{if } D(i_j, i_{j+1}) = \text{True} \implies \Big(O_j \cap O_{j+1} \neq \emptyset \ \text{ and } \forall o^* \in O_j \cap O_{j+1},\ s_j(o^*) = s^{\text{req}}_{j+1}(o^*) \Big)$$

where $D(i_j, i_{j+1})$ indicates a dependency from i_j to i_{j+1}.

Corollary. If there exists a pair (i_j, i_{j+1}) with a dependency such that $O_j \cap O_{j+1} = \emptyset$ or the state is inconsistent (i.e., $\exists o^* \in O_j \cap O_{j+1}$ such that $s_j(o^*) \neq s^{\text{req}}_{j+1}(o^*)$), then the sequence is invalid with respect to object dependencies.

Theorem 2 (Soundness of Sequential Composition)**.** *Let $i_1 = (a_1, o_1)$ and $i_2 = (a_2, o_2)$ be instructions. If*

$$\vdash i_1 \to_i i_2$$

is derivable using the deduction rules and object consistency holds, then for any state s,

$$\mathit{eval}(s, i_1 \to_i i_2) = S.$$

Proof. By the deduction rule, both i_1 and i_2 are derivable. By induction, $\text{eval}(s, i_1) = S$. Let $s' = \text{resultState}(s, i_1)$. By induction, $\text{eval}(s', i_2) = S$. Object consistency holds: the state of shared objects after i_1 matches the state before i_2. By the semantic clause for sequence,

$$\text{eval}(s, i_1 \to_i i_2) = S.$$

Thus, the rule is sound. □

Theorem 3 (Soundness of Functional Dependency)**.** *Let $i_1 = (a_1, o_1)$ and $i_2 = (a_2, o_2)$ be instructions. If*

$$\vdash i_1 \to_p i_2$$

is derivable using the deduction rules and functional dependency holds (state after i_1 equals state before i_2), then for any state s,

$$\mathit{eval}(s, i_1 \to_p i_2) = S.$$

Proof. By the deduction rule, both i_1 and i_2 are derivable. By induction, $\text{eval}(s, i_1) = S$. Functional dependency: $\text{resultState}(s, i_1) = s''$, and s'' is the state before i_2. By induction, $\text{eval}(s'', i_2) = S$. By the semantic clause for functional dependency,

$$\text{eval}(s, i_1 \to_p i_2) = S.$$

Thus, the rule is sound. □

Theorem 4 (Completeness of Sequential Composition)**.** *Let* $i_1 = (a_1, o_1)$ *and* $i_2 = (a_2, o_2)$ *be instructions. If for every state* s*,*

$$\mathit{eval}(s, i_1 \to_i i_2) = S,$$

and object consistency holds, then

$$\vdash i_1 \to_i i_2.$$

Proof. By the semantic clause for sequence,

$$\text{eval}(s, i_1 \to_i i_2) = S \implies \text{eval}(s, i_1) = S, \quad \text{eval}(s', i_2) = S,$$

where $s' = \text{resultState}(s, i_1)$, and object consistency holds.

By induction on the structure of instructions, since $\text{eval}(s, i_1) = S$ for all s, we have $\vdash i_1$.

Similarly, since $\text{eval}(s', i_2) = S$ for all s', we have $\vdash i_2$.

By the deduction rule for sequence composition, and object consistency, we conclude

$$\vdash i_1 \to_i i_2.$$

□

Theorem 5 (Completeness of Functional Dependency)**.** *Let* $i_1 = (a_1, o_1)$ *and* $i_2 = (a_2, o_2)$ *be instructions. If for every state* s*,*

$$\mathit{eval}(s, i_1 \to_p i_2) = S,$$

and functional dependency holds (state after i_1 *equals state before* i_2*), then*

$$\vdash i_1 \to_p i_2.$$

Proof. By the semantic clause for functional dependency,

$$\text{eval}(s, i_1 \to_p i_2) = S \implies \text{eval}(s, i_1) = S, \quad \text{eval}(s'', i_2) = S,$$

where $s'' = \text{resultState}(s, i_1)$, and functional dependency holds.

By induction on the structure of instructions, since $\text{eval}(s, i_1) = S$ for all s, we have $\vdash i_1$.

Similarly, since $\text{eval}(s'', i_2) = S$ for all s'', we have $\vdash i_2$.

By the deduction rule for functional dependency, and the state equality condition, we conclude

$$\vdash i_1 \to_p i_2.$$

□

6. Implementation

The proposed framework is computationally instantiated through the MIRA AI Agent, a system leveraging Large Language Models (e.g., Groq, Gemini) for instruction generation and sequence validation. This agent, detailed in future work, demonstrates the practical feasibility of our semantic model, with real-time deployment as a web application. Current efforts focus on formal verification, with implementation specifics to be elaborated in a forthcoming paper.