A MĪMĀṀSĀ INSPIRED FRAMEWORK FOR INSTRUCTION SEQUENCING

Abstract.

1. Introduction

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 or Preconditions
- $P = \{p_1, p_2, \ldots, p_n\}$ 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 2.1:

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

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:

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

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:

$$i_j = (a_j, o_j) \tag{2.2}$$

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.

Preprint submitted to
Logical Methods in Computer Science

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. **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:

$$\mathcal{M} = \langle \mathcal{R}, \mathcal{A}, \mathcal{O}, \mathcal{G}, intention, eval \rangle, \tag{2.3}$$

where

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

The semantic evaluations for different imperatives are given as follows:

- **Unconditional Imperatives**:

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

- **Imperative Enjoining Goal**:

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

- **Imperatives in Sequence**:

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

- **Imperatives in Parallel**:

$$eval((r_1, (a_1, o_1)) \wedge (r_2, (a_2, o_2))) = \begin{cases} S, & \text{if } eval(r_1, (a_1, o_1)) = S, \quad eval(r_2, (a_2, o_2)) = S, \\ & \text{and object consistency does not hold,} \\ V, & \text{if } eval(r_1, (a_1, o_1)) = V \text{ or } eval(r_2, (a_2, o_2)) = V. \end{cases}$$

- **Imperatives with Choice**:

$$eval((r_1, (a_1, o_1)) \oplus (r_2, (a_2, o_2))) = \begin{cases} S, & \text{if } eval(r_1, (a_1, o_1)) = S \text{ or } eval(r_2, (a_2, o_2)) = S, \\ V, & \text{if } eval(r_1, (a_1, o_1)) = V \text{ and } eval(r_2, (a_2, o_2)) = V. \end{cases}$$

This formal action-object representation provides a foundation for capturing the ordering and dependency relationships among instructions. The philosophy of Mīmāṁsā includes several principled methods for sequencing such instructions to achieve coherent execution.

The next section introduces these methods and formally develops three sequencing strategies by extending this representation.

3. 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 (*Pravriṭṭikrama*). 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 3.1, 3.2, and 3.3, respectively.

3.1. **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 Mīmāṁsā [**?**], let:

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

Then, Instruction in sequence can be expressed by Equation 3.1.

$$(a_t o_t) \to_i (a_{t+1} o_{t+1}) \tag{3.1}$$

where $\to_i$ 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, the temporal order and precedence can be formally represented using nested left brackets as shown in Equation 3.2.

$$(\cdots((a_1 o_1) \to_i (a_2 o_2)) \to_i (a_3 o_3) \cdots) \to_i (a_n o_n) \tag{3.2}$$

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:

$$(((pick\{rice\}) \to_i (cook\{rice, pot\})) \to_i (add\{rice, dish\})) \tag{3.3}$$

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.

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

This representation can be extended to a series of instructions as given by Equation 3.4.

$$(r_1 \rightarrow_r (i_1 \rightarrow_p p_1)), (r_2 \rightarrow_r (i_2 \rightarrow_p p_2), ..., (r_n \rightarrow_r (i_n \rightarrow_p p_n)) \tag{3.4}$$

If the purpose p_k of instruction i_k becomes the precondition 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 3.4 can be formalized as shown in Equation 3.5.

$$\forall j \in \{1, ..., n-1\} : r_j \rightarrow_r ((a_j, o_j) \rightarrow_p p_j), r_{j+1} = p_j \tag{3.5}$$

3.3. **Sequential and Parallel Execution Methods for Repetitive Tasks.** In some tasks, it is necessary to repeat the same action multiple times across different items. Two common approaches to sequence such repetitive actions have been identified in previous work with LLMs [?], reiterated here for clarity.

Consider the task of a teacher grading answer scripts from 20 students, each answer script containing 5 questions. In the **Sequential Completion Method**, the teacher grades all five questions of the first student's script before moving on to completely grade the second student's script, continuing this process sequentially for all students. Alternatively, the **Step-by-Step Parallel Method**, also known as the **Iterative Procedure**, involves the teacher grading the first question across all 20 scripts before proceeding to grade the second question for all scripts, and so forth.

3.3.1. *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 3.6.

$$(a_1 o_{1j} \rightarrow_i a_2 o_{2j} \rightarrow_i ... \rightarrow_i a_n o_{nj}) \tag{3.6}$$

This is repeated for all j as shown below.

$$\prod_{j=1}^{T} (a_1 o_{1j} \rightarrow_i \cdots \rightarrow_i a_n o_{nj}) \tag{3.7}$$

Equation 3.7 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.

3.3.2. *Step-by-Step Parallel Method (Iterative Procedure).* The step-by-step parallel method can be formalized using a parallel composition connective $\|_i$, which groups actions performed independently on different objects without enforcing temporal sequencing or object dependency among them.

Formally, this method is represented as follows:

$$\begin{aligned}\Big((a_1o_{11} \parallel_i a_1o_{12} \parallel_i \cdots \parallel_i a_1o_{1T}) \rightarrow_i \\ (a_2o_{21} \parallel_i a_2o_{22} \parallel_i \cdots \parallel_i a_2o_{2T}) \rightarrow_i \cdots \rightarrow_i \\ (a_no_{n1} \parallel_i a_no_{n2} \parallel_i \cdots \parallel_i a_no_{nT})\Big).\end{aligned} \tag{3.8}$$

Here:

- $\|_i$ groups the same action a_k applied to objects $o_{k1}, \ldots, o_{kT}$ in parallel, without a strict order or dependency between object-specific actions.
- $\rightarrow_i$ imposes a temporal ordering between these groups, requiring all actions in a given group to complete before the next group begins.

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.

In both the Sequential Completion and Step-by-Step Parallel Methods, object dependency is preserved in accordance with the principle of direct assertion (śrutikrama). Specifically, within the sequential composition $\rightarrow_i$, each instruction group or action sequence enforces the temporal order and dependency relations established by direct assertion, ensuring coherent and consistent execution even when tasks are performed repetitively over multiple objects.

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

4. Object Dependency and Functional Dependency in Instruction Sequencing

The validity of an instruction sequence relies on two main conditions: **object dependency** and **functional dependency** across instructions. These ensure that the instruction sequence respects the logical dependencies and state transformations of objects involved.

4.1. **Object Dependency Condition.**

Definition 1. ***Object Dependency Condition***
A sequence of instructions $I = \{i_1, i_2, \ldots, i_n\}$, where each $i_j = (a_j, O_j)$, exhibits valid object dependency if for every pair of consecutive instructions (i_j, i_{j+1}), there exists at least one object o^* such that:

$$o^* \in O_j \cap O_{j+1}.$$

This signifies that the two instructions share at least one object, establishing a valid dependency consistent with direct assertion (śrutikrama).

4.2. **Functional Dependency Condition.**

Definition 2. ***Functional Dependency Condition***
For instructions $i_j = (a_j, O_j)$ and $i_{j+1} = (a_{j+1}, O_{j+1})$ with shared objects $o^* \in O_j \cap O_{j+1}$, the sequence maintains functional dependency if:

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

Here:

- *$s_j(o^*)$ is the state of object o^* immediately after executing instruction i_j,*
- *$s_{j+1}^{\text{req}}(o^*)$ is the required state of o^* to start executing i_{j+1}.*

This relation captures the classical notion of functional dependency, where the post-state of an object after one instruction determines the input state required for the next.

4.3. **Validity Theorem for Sequencing.**

Theorem 1. ***Validity of Instruction Sequence with Object and Functional Dependencies***
Let $I = \{i_1, i_2, \ldots, i_n\}$ be a sequence of instructions with $i_j = (a_j, O_j)$. The sequence is valid if:

$$\forall j \in \{1, \ldots, n-1\}, \quad \textit{if} \quad D(i_j, i_{j+1}) = \textit{True},$$

then:

$$O_j \cap O_{j+1} \neq \emptyset \quad \textit{and} \quad \forall o^* \in O_j \cap O_{j+1}, \quad s_j(o^*) = s_{j+1}^{\text{req}}(o^*),$$

where $D(i_j, i_{j+1})$ expresses a dependency between i_j and i_{j+1}.

Proof. We prove the theorem by induction on the instruction sequence length, focusing on maintenance of functional dependency (state consistency) between consecutive instructions.

Base Case:

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

- Each object $o \in O_1$ is initially in state $s_1^{\text{init}}(o)$.
- There are no preceding instructions, so no dependencies are checked.
- Instruction i_1 is executable as long as its required preconditions on object states are satisfied by the 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 $1 \leq k \leq j-1$,

$$O_k \cap O_{k+1} \neq \emptyset \quad \text{and} \quad \forall o^* \in O_k \cap O_{k+1},\ s_k(o^*) = s_{k+1}^{\text{req}}(o^*)$$

That is, both object dependency and functional dependency are satisfied for all previous pairs in the sequence.

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

- **Dependency Check:** If $\exists o^* \in O_j \cap O_{j+1}$, then i_{j+1} depends on i_j through object o^*.
- **Functional Dependency Check:** The state of o^* after executing i_j, denoted $s_j(o^*)$, must match the required state for i_{j+1}, denoted $s_{j+1}^{\text{req}}(o^*)$:

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

This ensures that the functional dependency condition is preserved for (i_j, i_{j+1}), allowing i_{j+1} to be validly executed. By the principle of induction, the functional dependency holds for all pairs of instructions in the sequence, thereby establishing validity. $\square$

This theorem can be formalized as follows:

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

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

4.3.1. *Corollary.* A sequence is invalid if there exists a pair (i_j, i_{j+1}) such that:

- $O_j \cap O_{j+1} = \emptyset$, indicating no common object and thus no dependency.
- Or, there exists an object $o^* \in O_j \cap O_{j+1}$ for which:
$$s_j(o^*) \neq s_{j+1}^{\text{req}}(o^*),$$
violating the functional dependency condition.

4.4. **Deduction Rules for Sequencing.** Extending the original deduction rules from MIRA [**?**], this section provides a representation in terms of ¡*action,object*¿ pair towards the sequencing methods of direct assertion **??** and purpose based approach **??**. Among the deduction rules, only the rule conditional enjoining action *cni* with $X = \phi$, $Y = \psi$ and $w = i$ is applicable, which reflects the sequencing according to direct assertion method that includes object dependency condition. Additionally, the deduction rule of purpose based sequencing method is also included.

Object-Consistent Sequencing Rule (OCS). Assume i_1 leads to i_2 and there is at least one object common to both instructions. The deduction rule is:

$$\frac{[i_1] \quad \cdots \quad i_2 \qquad O_1 \cap O_2 \neq \emptyset O_1 \cap O_1 \neq \emptyset \quad \text{and} \quad \forall o^* \in O_1 \cap O_2}{i_1 \rightarrow_i i_2}\ \texttt{OCS}$$

Explanation: This rule states that the sequential composition $i_1 \rightarrow_i i_2$ is derivable if i_1 leads to i_2 and the two instructions share at least one object. This ensures that sequencing is only permitted when there is object overlap, reflecting the principle of object dependency.

Purpose-Linked Sequencing Rule (PLS). Let i_k and i_{k+1} be instructions with preconditions r_k, r_{k+1} and purposes p_k, p_{k+1}.

$$\frac{\vdash (r_k \rightarrow_r (i_k \rightarrow_p p_k)) \quad \vdash (r_{k+1} \rightarrow_r (i_{k+1} \rightarrow_p p_{k+1})) \quad \text{Functional dependency: } p_k = r_{k+1}}{\vdash (r_k \rightarrow_r (i_k \rightarrow_p p_k)) \rightarrow (r_{k+1} \rightarrow_r (i_{k+1} \rightarrow_p p_{k+1}))}\ PLS$$

This rule derives a sequenced instruction chain when both components are derivable and the purpose of the first instruction provides the precondition for the second (functional dependency). This implements purpose-based ordering in a set of given instructions.

These rules ensure that instruction sequencing is logically justified only when the relevant object and functional dependencies are maintained. They form the basis for the forthcoming proofs of soundness and completeness, which demonstrate the alignment of proof theory and semantic interpretation in this system.

5. Soundness and Completeness of Deduction Rules

In this section, we establish the soundness and completeness of the deduction rules (OCS and PLS) with respect to the semantic model and the Validity Theorem (Theorem 1). Soundness ensures that if a sequence of instructions is derivable using the rules, then it satisfies the semantic validity conditions (both object dependency and functional dependency). Completeness ensures the converse: if a sequence satisfies the semantic validity conditions, then it is derivable using the rules.

We consider sequences of instructions $I = \{i_1, i_2, \ldots, i_n\}$, where each $i_j = (a_j, O_j)$, and validity is defined per Theorem 1. The semantic model $M = \langle \mathcal{R}, A, O, G, \mathit{intention}, \mathit{eval} \rangle$ interprets these sequences, with satisfaction statuses S (satisfied), V (violated), or N (neutral). Derivability ($\vdash$) refers to proofs using OCS and PLS.

5.1. Soundness.

Theorem 2 (Soundness). If a sequence I is derivable ($\vdash I$) using OCS and PLS, then I is semantically valid in M (i.e., it satisfies Theorem 1: for all dependent consecutive pairs (i_j, i_{j+1}), $O_j \cap O_{j+1} \neq \emptyset$ and $\forall o^* \in O_j \cap O_{j+1}$, $r_j(o^*) = r^{\text{req}}_{j+1}(o^*)$, and $\mathit{eval}(r, I) = S$ for some initial precondition $r \in \mathcal{R}$).

Proof. We proceed by induction on the length of the derivation.

Base Case: For a single instruction $i_1 = (a_1, O_1)$, derivability is trivial (no rules applied). Semantic validity holds vacuously (no pairs). By the semantics, $\mathit{eval}(r, (a_1, O_1)) = S$ if a_1 is performed on objects in O_1 under precondition r, assuming initial preconditions match requirements.

Inductive Step: Assume soundness holds for derivations of length up to k. Consider a derivation of length $k + 1$. Since validity requires both object and functional dependencies, derivations involve applications of both OCS and PLS for sequences with such dependencies. The last rule applied could be OCS or PLS, but the full derivation combines them to capture both conditions.

- **Case OCS (Object Dependency):** The rule derives $i_1 \rightarrow_i i_2$ from premises $[i_1] \ldots i_2$, with $O_1 \cap O_2 \neq \emptyset$ and object consistency. By inductive hypothesis, the premises are semantically valid (each sub-sequence evaluates to S). Semantics for imperatives in sequence (Section 2.2) yield $\mathit{eval}(r, (a_1, o_1) \rightarrow_i (a_2, o_2)) = S$ if both sub-evaluations are S and object consistency holds. This satisfies the object dependency part of Theorem 1 for the pair. Functional dependency, if required, is preserved through concurrent or prior applications of PLS in the derivation.

- **Case PLS (Functional Dependency):** The rule derives $(r_k \rightarrow_r (i_k \rightarrow_p p_k)) \rightarrow (r_{k+1} \rightarrow_r (i_{k+1} \rightarrow_p p_{k+1}))$ from premises with functional dependency $p_k = r_{k+1}$. By inductive hypothesis, premises are valid. Semantics for purpose-enjoined imperatives ensure $\mathit{eval}(r, (a_k, o_k) \rightarrow_p g) = S$ only if intention holds and the action satisfies the goal. Specifically, in the context of functional dependency: (i) $(r_k \rightarrow_r (i_k \rightarrow_p p_k))$ evaluates to S if the intention of the goal p_k is true and the action i_k is performed with precondition r_k true; (ii) after execution, the intention of goal p_k becomes reality (i.e., the evaluation from step (i) turns to a true value, serving as the precondition for the next step r_{k+1}); (iii) the next $(r_{k+1} \rightarrow_r (i_{k+1} \rightarrow_p p_{k+1}))$ then becomes S when the intention of goal p_{k+1} is true and action i_{k+1} is performed with precondition r_{k+1}. The linkage $p_k = r_{k+1}$ enforces functional dependency (post-precondition of i_k provides the precondition for i_{k+1}), yielding overall S. Object dependency, if required, is enforced through concurrent or prior OCS applications in the full derivation (e.g., ensuring shared objects with consistent preconditions), satisfying both parts of Theorem 1.

Derivations combine OCS and PLS as needed (e.g., OCS for object overlap and PLS for purpose-precondition chaining in mixed sequences), ensuring both dependencies are captured and leading to overall semantic validity by induction.

5.2. **Completeness. Theorem 3 (Completeness).** If a sequence I is semantically valid in M (i.e., satisfies Theorem 1 and $eval(r, I) = S$ for some $r \in \mathcal{R}$), then I is derivable ($\vdash I$) using OCS and PLS.

Proof. Again, by induction on the sequence length n.

Base Case: For $n = 1$, i_1 is valid if $eval(r, (a_1, O_1)) = S$. No rules needed; derivability is trivial.

Inductive Step: Assume completeness for sequences up to length $n - 1$. For length n, since I is valid, all consecutive pairs satisfy both object and functional dependencies per Theorem 1.

The sub-sequence $\{i_1, \ldots, i_{n-1}\}$ is valid by assumption, so derivable by inductive hypothesis. To extend to the full sequence, apply both OCS and PLS as required by the dependencies:

- Apply OCS for object dependency (pairs share objects $O_j \cap O_{j+1} \neq \emptyset$ and preconditions match), deriving temporal sequencing $i_{n-1} \rightarrow_i i_n$.

- Apply PLS for functional dependency (pairs link via $p_j = r_{j+1}$), deriving purpose-chained sequencing, where the semantic evaluation to S follows the steps: (i) intention of p_{n-1} true and action performed under r_{n-1}; (ii) post-execution realization of p_{n-1} as true, becoming r_n; (iii) subsequent evaluation to S for the next imperative under r_n with intention of p_n true.

Since Theorem 1 requires both dependencies for validity, apply both rules for each relevant pair: OCS to enforce object overlap and precondition consistency, and PLS to enforce purpose-precondition linkage. For mixed sequences, decompose into sub-sequences, derive them using the appropriate rule(s), then combine. Semantic validity ensures both conditions align with the premises of OCS and PLS, so the full derivation exists by combining the rules.

By induction, all semantically valid sequences are derivable.

These proofs align the proof theory (deduction rules) with the semantics, ensuring the framework's logical consistency for instruction sequencing.

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.