A MĪMĀṀSĀ INSPIRED FRAMEWORK FOR INSTRUCTION SEQUENCING

Abstract.

1. Introduction

In the rapidly evolving field of artificial intelligence, AI agents are increasingly tasked with executing complex sequences of instructions to achieve specified goals. These agents, ranging from robotic systems to large language model (LLM)-based assistants, must interpret, order, and perform actions in a coherent manner to ensure effective and reliable outcomes. However, challenges arise in handling ambiguous, interdependent, or repetitive instructions, where improper sequencing can lead to inefficiencies, errors, or failures. Traditional approaches in task planning and imperative logic often fall short in addressing these issues, particularly in dynamic environments where dependencies between actions and objects evolve over time.

This paper draws inspiration from the ancient Indian philosophical system of Mīmāṁsā, a hermeneutical tradition focused on interpreting Vedic texts through principled methods of ordering and resolving ambiguities in prescriptions. Mīmāṁsā offers a rich set of sequencing principles—such as Śrutikrama (direct assertion), Arthakrama (purpose-based ordering), and Pravṛttikrama (iterative procedure)—that ensure uninterrupted and coherent execution. By adapting these principles to computational frameworks, we propose an extended formal system for instruction sequencing in AI agents.

Building upon the foundational work of Srinivasan and Parthasarathi (2021) on imperative logic with action-object pairs, this framework refines the syntax and semantics to explicitly incorporate preconditions, purposes, and dependencies. Key contributions include:

- An extension of the imperative language to decompose instructions into action-object pairs, enabling precise representation and evaluation.
- Formalization of three Mīmāṃsā-inspired sequencing methods: direct assertion for temporal ordering, purpose-based sequencing for functional dependencies, and iterative procedures for repetitive tasks.
- Definitions of object dependency and functional dependency conditions, along with a validity theorem that ensures sequence coherence through state consistency.
- Deduction rules (Object-Consistent Sequencing and Purpose-Linked Sequencing) for deriving valid sequences, with proofs of soundness and completeness relative to the semantic model.

Preprint submitted to
Logical Methods in Computer Science

- Discussion of implementation via the MIRA AI Agent, leveraging LLMs for practical validation.

The remainder of the paper is structured as follows: Section 2 extends the syntax and semantics for action-object imperative logic. Section 3 introduces the sequencing methods. Section 4 details object and functional dependencies, including the validity theorem. Section 5 provides the deduction rules and proofs of soundness and completeness. Section 6 outlines the implementation, followed by conclusions.

2. Related Work

Reasoning with actions has been the focus of extensive research across multiple domains, including formal logic, task planning and more recently, large language models. This section presents an overview of the major developments in these areas.

2.1. **Classical Planning and Temoral Logic.** Traditional planning systems, such as STRIPS, model actions as transitions between states [Fikes and Nilsson, 1971]. Each action is described using classical logic formulas that define its preconditions and effects. A search process is employed to navigate from the initial state—characterized by the preconditions—to the goal state—characterized by the desired effects. The resulting path represents a sequence of actions that together form a plan.

Although this method is robust, there are some disadvantages:

- Simultaneous actions cannot be represented
- Representation of composite actions is difficult
- External event representation is not possible

To some extent, these challenges have been mitigated by incorporating temporal logic into planning frameworks [Allen, 1983]. Temporal logic deals with reasoning about time using the operators $\Box$ (*always*) and $\Diamond$ (*eventually*), $\bigcirc$ (*next*) and U (*Until*) [Huth and Ryan, 2004] In this approach, events are treated as objects, and necessary conditions—such as preconditions—are expressed over temporal intervals. This allows for the construction of non-linear plans and provides the flexibility to represent concurrent actions. However, this method does not handle the decomposition of complex plans into simpler tasks. To overcome this limitation, a hierarchical planning structure is adopted, incorporating temporal reasoning across subtasks [Song and Cohen, 1991]. Extending the work of Allen, Henry A. Kautz and Peter B. Ladkin, has provided a temporal reasoning system that handles dates, duration and disjointness [Kautz and Ladkin, 1991].

Given the general intractability of AI planning, one effective approach is to incorporate domain-specific knowledge. In this direction, knowledge specific to the problem domain is utilized to guide the search process, resulting in a sequence of actions that constitute a valid plan [**?**]. This domain knowledge is expressed using Linear Temporal Logic (LTL), along with the GOAL modality. This method offers several advantages, including modularity, a declarative specification of the planning problem, and the transformation of intractable problems into solvable ones. In another work, the goal is represented as a sequence of states rather than a single final state [Bacchus and Kabanza, 1996]. Metric Interval Temporal Logic (MITL) is employed to capture various types of goals, including those with temporal deadlines, quantified goals (both universal and existential), as well as safety and maintenance goals.

Recent advancements in automated planning have highlighted various approaches to explanation-based reasoning [Chakraborti et al., 2020]. These approaches encompass algorithmic methods, model-based techniques, inference reconciliation, model reconciliation, and plan-based explanations.

2.2. **Hierarchial Planning and Task Networks.**

2.3. **Plan Recognition and Goal Inference.**

2.4. **Dependency Analysis and Constraint Satisfaction.**

2.5. **Multi-Agent Coordination and Distributed Planning.**

2.6. **BDI Agents and Practical Reasoning.**

2.7. **Reactive Planning and Dynamic Environments.**

2.8. **Formal Methods and Verification.**

2.9. **Normative Systems and Philosophical Foundations.**

2.10. **Business Process Modeling and Workflow Management.**

2.11. **Imperative Logic and Action Representation.** Imperatives differ from declarative statements in that they cannot be evaluated as *True* or *False*, which limits the applicability of classical logic in reasoning about them. To address this, several studies have explored translating imperatives into propositions and then applying classical logic rules. However, such approaches often encounter paradoxes, as highlighted by Ross, revealing fundamental limitations in representing imperatives within standard logical frameworks. A detailed survey of various perspectives on imperative logic, particularly in the context of action representation in AI, has been presented in [Srinivasan and Parthasarathi, 2017].Inspired by the Indian philosophical system of Mīmāṁsā, a three-valued formalism has been proposed to handle imperatives, where each instruction is evaluated as *Satisfaction* (S), *Violation* (V), or *No intention to achieve the goal* (N) [Srinivasan and Parthasarathi, 2021]. This framework provides a more expressive representation of the normative and intentional aspects of actions than classical logic. Building on this foundation, the same authors have also extended the formalism to handle the temporal sequencing of instructions, enabling reasoning about ordered action execution [Srinivasan and Vijayan, 2025].

2.12. **Task Planning with Large Language Models.** Oflate LLM has been seen to be helpful for helping task planning. Although LLMs excel at text generation and general reasoning, they struggle with temporal reasoning. A proposed solution involves translating temporal information into a graph structure, enabling LLMs to learn and reason over it more effectively [Xiong et al., 2024]. LLMs do not always produce truthful or consistent responses and may sometimes generate hallucinated or incorrect information. To address these issues and better align model outputs with human expectations, InstructGPT was developed using reinforcement learning from human feedback (RLHF) [Ouyang et al., 2022]

Positioning of the proposed paper. The proposed Mīmāṁsā inspired framework contributes to this landscape by providing a philosophically grounded approach to instruction sequencing that addresses several gaps in existing work:

- Philosophical Foundation: Unlike purely algorithmic approaches, this framework draws on sophisticated hermeneutical principles from Mīmāṁsā philosophy, providing principled methods for resolving ambiguities and establishing coherent sequences.
- Action-Object Decomposition: The explicit representation of instructions as action-object pairs enables more precise dependency analysis than traditional atomic action representations.
- Multiple Sequencing Paradigms: The formalization of three complementary sequencing methods (direct assertion, purpose-based, and iterative procedure) provides flexibility for different types of instruction coordination challenges.
- Formal Verification: Our deduction rules and completeness proofs provide formal guarantees about sequence validity, bridging the gap between philosophical principles and computational verification.
- Cultural Perspective: By drawing from Indian philosophical traditions, our work expands the Western-centric theoretical foundations of AI planning and reasoning systems.

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

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

3.1. **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 3.1:

$$\mathcal{F}_i = \{i \mid i \rightarrow_p p \mid (i \rightarrow_p p_1) \wedge (j \rightarrow_p p_2) \mid (i \rightarrow_p \theta) \oplus (j \rightarrow_p \theta) \mid (\varphi \rightarrow_i \psi) \mid (\tau \rightarrow_r \varphi)\} \quad (3.1)$$

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:

$$i_j = (a_j, o_j) \tag{3.2}$$

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 3.1.

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.

3.2. **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:

$$\mathcal{M} = \langle \mathcal{R}, \mathcal{A}, \mathcal{O}, \mathcal{G}, intention, eval \rangle, \tag{3.3}$$

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**:

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

• **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.

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 (*Pravr̤ttikrama*). 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 4.3, respectively.

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 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 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, the temporal order and precedence can be formally represented using nested left brackets as shown in Equation 4.2.

$$(\cdots((a_1 o_1)\rightarrow_i (a_2 o_2))\rightarrow_i (a_3 o_3)\cdots)\rightarrow_i (a_n o_n) \tag{4.2}$$

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 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 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 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 4.4 can be formalized as shown in Equation 4.5.

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

4.3. **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.

4.3.1. *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 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.

$$\prod_{j=1}^{T} (a_1 o_{1j} \rightarrow_i \cdots \rightarrow_i a_n o_{nj}) \tag{4.7}$$

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.

4.3.2. *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:

$$\begin{aligned} \Big(& (a_1 o_{11} \parallel_i a_1 o_{12} \parallel_i \cdots \parallel_i a_1 o_{1T}) \rightarrow_i \\ & (a_2 o_{21} \parallel_i a_2 o_{22} \parallel_i \cdots \parallel_i a_2 o_{2T}) \rightarrow_i \cdots \rightarrow_i \\ & (a_n o_{n1} \parallel_i a_n o_{n2} \parallel_i \cdots \parallel_i a_n o_{nT}) \Big). \end{aligned} \tag{4.8}$$

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.

5. 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.

5.1. 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).

5.2. 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.

5.3. 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. □

This theorem can be formalized as follows:

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

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

5.3.1. *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.

5.4. **Deduction Rules for Sequencing.** Extending the original deduction rules from MIRA Srinivasan and Parthasarathi [2021], 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:

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

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.

6. 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 $M = \langle \mathcal{R}, A, O, G, \textit{intention}, \textit{eval} \rangle$ interprets these sequences, with satisfaction statuses S (satisfied), V (violated), or N (neutral). Derivability ($\vdash$) refers to proofs using OCS and PLS.

6.1. 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 $\forall o^* \in O_j \cap O_{j+1}$, $r_j(o^*) = r_{j+1}^{\text{req}}(o^*)$, and $\textit{eval}(r, I) = S$ 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, $\textit{eval}(r, (a_1, O_1)) = S$ 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 $[i_1] \dots i_2$, 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.

6.2. **Completeness. Theorem 3 (Completeness).** If a sequence I is semantically valid in M (i.e., satisfies Theorem 1 and $\mathit{eval}(r, I) = S$ 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 $\mathit{eval}(r, (a_1, O_1)) = S$. 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 $\{i_1, \dots, i_{n-1}\}$ 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.

7. 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.

References

James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 2, pages 1215–1222. AAAI Press, 1996.

Tathagata Chakraborti, Sarath Sreedharan, and Subbarao Kambhampati. The emerging landscape of explainable automated planning & decision making. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20)*, pages 4803–4811. AAAI Press, 2020. URL `https://www.ijcai.org/proceedings/2020/bibtex/669`.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2 edition, 2004.

Henry A. Kautz and Peter B. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 1, pages 241–246, Anaheim, California, 1991. AAAI Press.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL `https://arxiv.org/abs/2203.02155`.

Fei Song and Robin Cohen. Temporal reasoning during plan recognition. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 1*, AAAI'91, page 247–252. AAAI Press, 1991. ISBN 0262510596.

Bama Srinivasan and Ranjani Parthasarathi. A survey of imperatives and action representation formalisms. *Artif. Intell. Rev.*, 48(2):263–297, August 2017.

Bama Srinivasan and Ranjani Parthasarathi. A formalism to specify unambiguous instructions inspired by mīmāṃsā in computational settings. *Logica Universalis*, 16(1):27–55, 2021.

Bama Srinivasan and Mohan Raj Vijayan. A Mīmāṃṣā inspired framework towards temporal reasoning in large language models. In *Proceedings of the 11th Indian Conference on Logic and Applications*, Calcutta, India, 2025. Springer LNCS.

Siheng Xiong, Ali Payani, Ramana Kompella, and Faramarz Fekri. Large language models can learn temporal reasoning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10452–10470, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.563. URL https://aclanthology.org/2024.acl-long.563/.