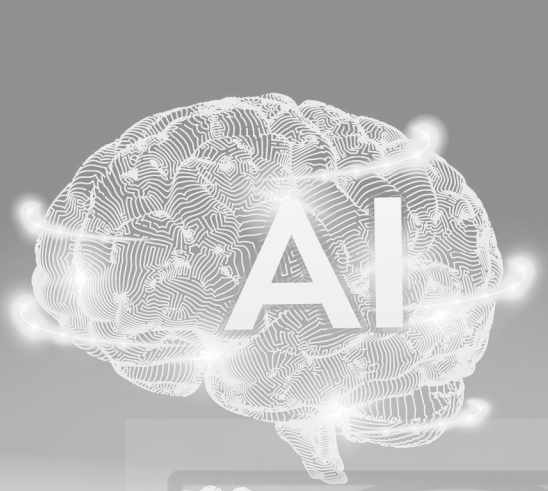

PART II

Knowledge Representation, Reasoning and Planning

Chapters Included

10 Knowledge Representation

One of the defining characteristics that distinguishes Artificial Intelligence (AI) systems from conventional computational programs is their ability to *represent knowledge explicitly*. While many AI techniques rely on search, optimization or numerical computation to produce solutions, truly intelligent behavior requires something more fundamental—a structured internal model of the world about which the system can reason. Knowledge representation addresses this requirement by providing formal means to encode information about the environment, objects, actions, and relationships in a form that a machine can interpret and manipulate.

Search-based techniques explore spaces of possible solutions by systematically examining alternative states or configurations. Such approaches are powerful, but they are inherently limited when the problem domain is vast, abstract or governed by general principles rather than enumerated possibilities. In contrast, knowledge representation focuses on how information itself is organized—how facts are stored, how rules capture regularities, and how relationships express structure. By operating on symbolic descriptions rather than raw states alone, an intelligent system gains the ability to draw inferences, generalize from known information, and reason about situations it has not encountered explicitly.

In classical AI, intelligence is not viewed merely as the execution of algorithms or the performance of calculations. Instead, it is understood as the deliberate manipulation of symbolic representations that stand for real-world entities, properties, relationships, and laws. Symbols serve as abstractions that allow an AI system to reason about the world without direct interaction at every step. Through these abstractions, complex phenomena can be reduced to logical structures that support explanation, prediction, and decision-making.

This chapter introduces the foundational ideas and formal tools used to represent knowledge in a precise and machine-interpretable form. It examines why explicit representations are necessary, how logical languages provide rigor and clarity, and how knowledge bases serve as the core repositories of information for intelligent systems. The representations developed here form the conceptual substrate upon which reasoning, planning, and learning mechanisms operate. As such, knowledge representation occupies a central position in classical AI, linking perception and problem-solving to inference, deliberate action, and adaptive behavior in the chapters that follow.

10.1 WHY KNOWLEDGE REPRESENTATION IS NEEDED

Problem-solving techniques such as state-space search and constraint satisfaction form the backbone of many classical AI systems. These methods explore spaces of states or variable assignments to find solutions that satisfy goals or constraints. When a domain is finite, well-defined, and enumerable—as in puzzles, games or scheduling—such techniques are effective and elegant.

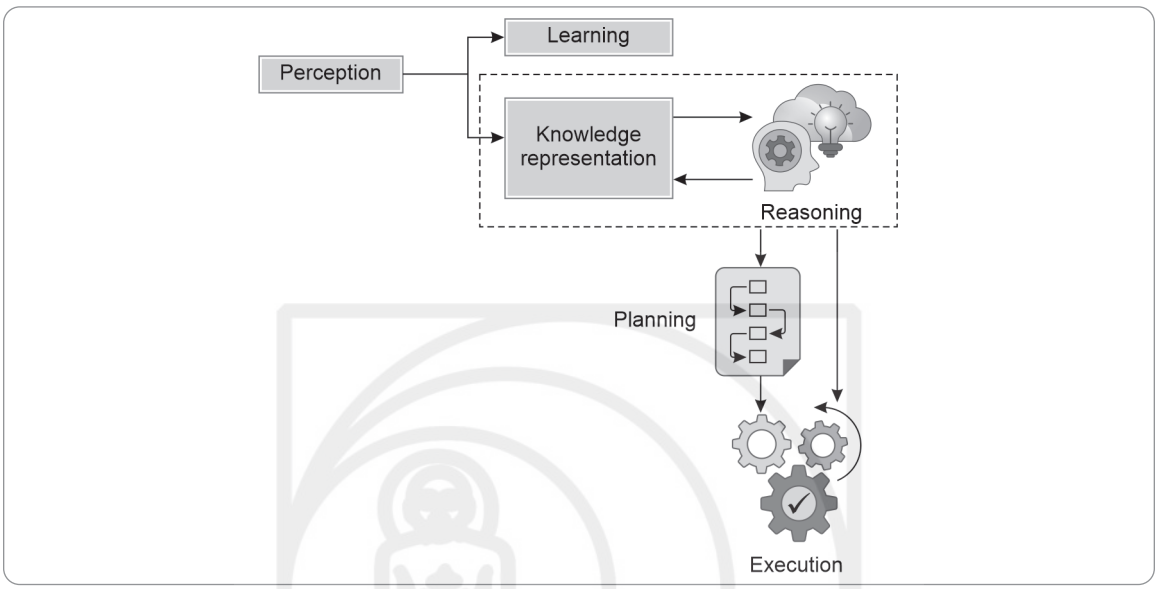

Fig. 10.1: Knowledge representation as the central structure linking perception, learning, reasoning, planning, and execution in an intelligent system

However, realistic domains are rarely exhaustively enumerable. The world is large, partially observable, dynamic, and governed by general principles rather than explicitly listed possibilities. Intelligent behavior therefore requires reasoning about *facts*, *relationships*, *rules*, and *abstract concepts* that apply across many situations.

Figure 10.1 illustrates the central role of knowledge representation within an intelligent system. Perception supplies information from the environment; learning refines internal models; knowledge representation organizes structured domain knowledge; reasoning operates over that knowledge; planning determines courses of action; and execution interacts with the world. Knowledge representation serves as the core structure that enables reasoning and planning to operate meaningfully.

Knowledge representation enables a system to move from *enumeration* to *structured understanding*. Instead of treating situations independently, the system stores general knowledge and applies it to new contexts. This capability is essential for intelligent reasoning and decision-making.

More specifically, knowledge representation is required for:

- **Scale and complexity:** Real-world environments contain many entities and interactions. Enumerating all possible states is computationally infeasible. Explicit representations allow reasoning to focus only on relevant knowledge.
- **General and abstract principles:** Intelligent behavior depends on reusable generalizations (e.g., 'all birds have wings' or 'bacterial infections may require antibiotics'). Representations allow such knowledge to be expressed compactly and applied uniformly.
- **Inference beyond stored facts:** Intelligence requires deriving implicit consequences. If a system knows that all metals conduct electricity and that copper is a metal, it should infer that copper conducts electricity. Such inference is possible only when knowledge is represented in a form that makes structure and relations explicit.

A well-designed knowledge representation enables an AI system to:
1. Store knowledge in a structured, reusable form,
2. Derive conclusions and answer queries through reasoning,
3. Provide transparent explanations of its decisions,
4. Update beliefs in response to new information.

10.2 TYPES OF KNOWLEDGE IN INTELLIGENT SYSTEMS

Intelligent behavior does not arise from a single uniform body of knowledge, but from the coordinated use of multiple complementary forms, each serving a distinct functional role within an AI system. Classical AI emphasizes the *explicit representation* of these forms using symbolic and interpretable structures, so that reasoning can be analyzed, conclusions explained, and behavior understood in principled terms.

Unlike approaches based primarily on implicit statistical associations, classical AI treats knowledge as an explicit computational object. It is stored, queried, manipulated, and transformed through formal representations such as logical expressions, rules, graphs, and plans. This explicitness supports justification of actions and systematic belief revision.

Different components of an intelligent system rely on different types of knowledge. Some encode *facts* about entities, properties, and relationships; others encode *procedures* for achieving goals. Additional forms provide *heuristic guidance* to improve efficiency, while *structural knowledge* organizes concepts into hierarchies and dependency patterns.

These distinctions are functionally significant. Confusing factual content with procedural control or lacking structural organization, leads to inefficiency and brittleness. By separating and coordinating knowledge types, classical AI achieves modularity, clarity, and flexibility in reasoning and action. Figure 10.2 presents classifications of knowledge types.

10.2.1 Declarative Knowledge

Declarative knowledge concerns *what is true* about the world. It captures facts, properties, relationships, and general laws describing a domain, without specifying how that knowledge is used computationally. It characterizes states of affairs and the regularities that govern them.

Declarative knowledge answers questions such as—What entities exist? What attributes do they possess? How are they related? What constraints hold universally? It provides the conceptual vocabulary for reasoning.

In classical AI, declarative knowledge is typically represented in formal languages such as propositional and first-order logic, whose precise syntax and semantics support unambiguous interpretation and sound inference.

Examples: A factual assertion:

Capital(*Delhi, India*),

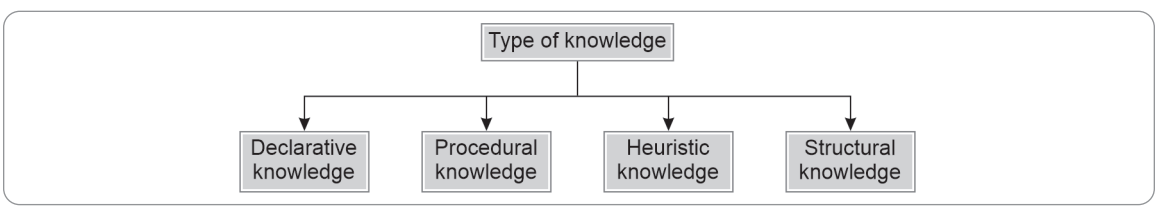

Fig. 10.2: Classification of knowledge types

or a general rule:

$$\forall x\,(Human(x) \rightarrow Mortal(x)).$$

Declarative knowledge may also encode constraints:

$$\forall x\,\neg(Student(x) \wedge Staff(x)),$$

ensuring consistency within the knowledge base.

10.2.2 Procedural Knowledge

Procedural knowledge concerns *how to perform tasks* or achieve goals. Where declarative knowledge specifies what holds, procedural knowledge specifies what to do and how to do it. It is action-oriented and typically expressed as algorithms, control strategies or execution rules.

It answers questions such as—What steps accomplish a task? In what order should actions occur? Under what conditions should alternatives be selected?

Examples: Procedural knowledge includes:

- A sorting algorithm
- A navigation strategy for obstacle avoidance
- A plan consisting of actions that achieves a goal state

In planning and search, it governs state expansion, action application, and solution construction. Production systems exemplify procedural knowledge through rules of the form:

```
IF condition THEN action
```

which directly connect reasoning to behavior.

10.2.3 Heuristic Knowledge

Heuristic knowledge consists of problem-specific guidance that improves efficiency. Unlike declarative truths or procedural mechanisms, heuristics suggest which alternatives are likely to be more promising. They aim to reduce computational effort rather than guarantee correctness.

In large or exponential search spaces, heuristics focus reasoning on relevant states or actions.

Examples: Typical heuristics include:

- Prefer actions that reduce distance to the goal.
- Choose the most constrained variable first.
- Expand the node with lowest estimated cost in A* search.

Heuristics encode experiential or structural insight. They are context-dependent and may succeed or fail depending on the domain, distinguishing them from universally valid declarative laws.

10.2.4 Structural Knowledge

Structural knowledge concerns how knowledge within a domain is organized. It captures relationships among concepts, levels of abstraction, and dependency patterns. Rather than describing isolated facts, it defines the architecture of the domain.

This organization enables abstraction, regularity exploitation, and complexity management.

Common forms: Structural knowledge appears as:

- Taxonomies and class hierarchies
- Graphs representing relations or constraints

- Trees structuring decisions or classifications.
- Probabilistic networks encoding dependencies.

Example: If *Student* is a subclass of *Person*, then every student inherits properties of persons. Knowledge defined at the higher level need not be repeated, ensuring economy and consistency.

10.2.5 Interplay Among Knowledge Types

In practical intelligent systems, knowledge types do not operate independently; intelligent behavior arises from their integration. Declarative knowledge provides the factual foundation, procedural knowledge enables execution, heuristic knowledge improves efficiency, and structural knowledge organizes domain complexity. Coherent, goal-directed intelligence depends on their coordinated use. Declarative knowledge specifies what is true about the domain. Procedural knowledge determines how actions are carried out. Heuristic knowledge guides search toward promising alternatives. Structural knowledge shapes the organization of concepts and states. Each contributes a distinct but complementary function.

Illustrative example—planning systems: A classical planning system illustrates this interaction. Declarative knowledge represents facts, objects, and action preconditions and effects. Procedural knowledge governs state transitions and plan execution. Heuristic knowledge guides search (e.g., goal-distance estimates). Structural knowledge represents states and actions within a graph structure.

Complementarity: These forms are mutually dependent. Declarative content without procedures cannot drive action; procedures without declarative grounding lack meaning. Heuristics require structure to be effective, and structure without content provides no basis for reasoning. Only their integration enables systematic reasoning and action.

10.3 KNOWLEDGE BASES

A *knowledge base* is a formal repository of knowledge used by an intelligent system to support reasoning and decision-making. Rather than embedding domain-specific information directly into procedural code, classical AI systems store knowledge declaratively in a knowledge base and apply general-purpose inference procedures to reason with it.

A knowledge base provides a structured and formal mechanism for storing and reasoning with knowledge. Its declarative nature, separation from inference, and logical foundations make it a central component of classical AI. Knowledge bases serve as the substrate upon which reasoning, planning, and learning mechanisms operate, forming a crucial bridge between symbolic representation and intelligent behavior.

At a fundamental level, a knowledge base consists of two components:

1. A set of sentences expressed in a formal knowledge representation language (such as propositional or first-order logic).
2. An inference mechanism that derives new sentences from existing ones according to the rules of logic.

This architecture allows intelligent behavior to emerge from the interaction between stored knowledge and reasoning processes.

10.3.1 Declarative Nature of Knowledge

Knowledge bases store information in a *declarative* form, meaning that they explicitly describe *what is true* about the world rather than prescribing *how* computations should be carried out. This distinction between *declarative knowledge* and *procedural control* is one of the most important conceptual advances introduced by classical AI.

In a procedural system, knowledge is embedded implicitly within the control flow of the program. For example, determining whether an individual is mortal might require explicit conditional checks, nested loops, and hard-coded rules. Such programs intertwine domain knowledge with algorithmic logic, making them difficult to extend, reuse or explain.

In contrast, a declarative knowledge base represents domain facts and general rules as logical sentences. For instance, the following sentences may be stored in a knowledge base:

$$Human(Socrates),$$

$$\forall x\,(Human(x) \rightarrow Mortal(x)).$$

These statements do not specify any algorithm. They merely assert facts and relationships that are assumed to hold in the domain. An inference mechanism—such as logical deduction or resolution—can then operate on these sentences to derive new conclusions. From the two sentences above, the system can logically infer:

$$Mortal(Socrates).$$

Crucially, this conclusion is not explicitly stored; it is obtained purely through reasoning. If additional facts are introduced, such as:

$$Human(Vyāsa),$$

the same general rule immediately yields:

$$Mortal(Vyāsa),$$

without requiring any modification to the inference procedure.

This separation between *knowledge* and *reasoning* provides several important advantages:

- **Modularity:** Domain knowledge can be added, removed or modified without altering the reasoning algorithm.
- **Reusability:** The same inference mechanism can be applied to different knowledge bases across diverse domains.
- **Explainability:** Conclusions can be justified by tracing them back to explicit facts and rules stored in the knowledge base.
- **Scalability:** General rules apply uniformly to new objects as the domain grows.

From a theoretical perspective, the declarative nature of knowledge bases aligns closely with the foundations of logic and discrete mathematics, where meaning is assigned through semantics rather than through execution order. From a practical AI perspective, it enables systems to reason in a transparent, principled, and extensible manner.

Thus, declarative knowledge representation forms the cornerstone of knowledge-based AI systems, allowing intelligent behavior to emerge from logical inference rather than from rigid procedural control.

10.3.2 Separation of Knowledge and Inference

A central and defining design principle in classical AI is the strict separation between *knowledge* and *inference*. Knowledge refers to the declarative information stored about a domain, while inference refers to the general reasoning procedures used to derive new conclusions from that information. By keeping these two aspects distinct, AI systems achieve clarity, flexibility, and robustness.

In such an architecture, a knowledge base contains domain-specific facts and rules expressed in a formal language, whereas the inference engine implements domain-independent logical operations such as deduction, resolution or chaining. As a result, the same inference mechanism can operate on entirely different knowledge bases, and conversely, the same knowledge base can be queried using different inference strategies depending on the task at hand.

Illustrative example: Consider a knowledge base containing the following statements:

$$Human(Vyāsa),$$

$$\forall x\,(Human(x) \rightarrow Mortal(x)).$$

An inference engine applying standard rules of first-order logic can derive:

$$Mortal(Vyāsa).$$

If the knowledge base is later extended with new facts, such as:

$$Human(Pāṇini),$$

the same inference engine immediately derives:

$$Mortal(Pāṇini),$$

without requiring any modification to the reasoning algorithm itself. The inference procedure remains unchanged; only the knowledge base evolves.

Advantages of this separation: The separation of knowledge and inference offers several important benefits:

- **Modifiability of knowledge:** Domain knowledge can be updated, expanded or corrected simply by adding or removing sentences from the knowledge base, without altering the inference mechanism. This is especially valuable in dynamic or evolving domains.
- **Reusability of inference mechanisms:** A single inference engine can be reused across multiple applications, such as medical diagnosis, legal reasoning or robotic planning, by supplying different knowledge bases. This reduces duplication of effort and promotes systematic system design.
- **Explainability and transparency:** Because conclusions are derived through explicit logical rules from explicitly stated facts, the system can provide clear explanations for its decisions. Each conclusion can be traced back to the premises and inference rules that produced it.
- **Theoretical soundness:** The separation mirrors the structure of formal logic, where axioms (knowledge) are distinct from rules of inference. This alignment with mathematical foundations enables rigorous analysis of correctness, soundness, and completeness.

This principle is foundational to many classical AI architectures, including expert systems, logic programming languages, and rule-based agents. In expert systems, for example, the knowledge base encodes expert domain knowledge, while the inference engine performs reasoning that simulates expert decision-making. Similarly, in logic programming, programs are treated as sets of logical sentences, and computation is viewed as logical inference.

10.3.3 Structure of a Logical Knowledge Base

In logical AI systems, a knowledge base is not an unstructured collection of statements but a carefully organized set of logical sentences that together describe a domain. From a formal perspective, a logical knowledge base can be viewed as a finite set of well-formed formulas in a given logical language. These formulas play different conceptual roles within the overall representation.

A typical logical knowledge base contains the following components:

- **Facts:** Facts are atomic sentences that describe specific, concrete truths about the current state of the world. They do not involve implication or quantification beyond what is needed to name objects. Examples include:

  $$Student(Alice), \qquad At(Robot1, RoomA).$$

  Here, $Student(Alice)$ asserts that a particular individual named *Alice* is a student, while $At(Robot1, RoomA)$ specifies the current location of a robot. Facts serve as the foundational evidence upon which reasoning is performed.

- **Rules:** Rules are general logical implications that express domain laws, regularities or causal relationships. They typically involve variables and quantifiers and apply to many objects at once. A common example is:

$$\forall x\ (Student(x) \rightarrow Registered(x)).$$

  This rule captures a general principle of the domain—every student must be registered. Rules allow the knowledge base to generalize beyond explicitly stated facts and enable the inference of new information.
- **Constraints:** Constraints are logical sentences intended to restrict the class of admissible models in a valid interpretation of the domain. They restrict the set of allowable worlds by ruling out logically inconsistent or impossible situations. For example:

$$\forall x\ \neg(Student(x) \wedge Staff(x)).$$

This constraint states that no individual can simultaneously be both a student and a staff member. Constraints play a critical role in maintaining consistency within the knowledge base and in detecting contradictions during reasoning.

Together, facts, rules, and constraints form a coherent and expressive model of the domain. Facts describe what is currently known, rules describe what must follow from that knowledge, and constraints define what is logically permissible. An inference mechanism operating over this structured knowledge base can derive new conclusions, detect inconsistencies, and answer queries in a principled and logically sound manner.

From a discrete mathematics viewpoint, this structure mirrors the axiomatic method, where axioms (facts and rules) define a theory and constraints limit its models. From an AI perspective, it provides a clear and modular foundation for reasoning, planning, and decision-making in knowledge-based systems.

10.3.4 Queries and Entailment

The fundamental operation performed on a knowledge base is the answering of *queries*. A query represents a formal question posed to an intelligent system about the domain it models. From a conceptual standpoint, querying a knowledge base is the act of asking whether a particular statement is a logical consequence of the knowledge already stored. The role of the inference mechanism is therefore not to generate new facts arbitrarily, but to determine whether the information encoded in the knowledge base *supports* the query in a logically sound manner.

A query may take many forms depending on the underlying representation language. In propositional logic, a query is typically a well-formed formula, such as a conjunction or implication. In first-order logic, a query may involve variables, predicates, and quantifiers, often requesting the existence of objects that satisfy certain conditions. Regardless of its form, the central question remains the same: *Does the knowledge base logically imply the query?*

Semantic definition of entailment: Formally, let *KB* denote a knowledge base and let α be a sentence expressed in the same logical language. We write:

$$KB \models \alpha$$

to denote that α is *entailed* by *KB*. Entailment means that for every interpretation (model) $\mathcal{M}$, if $\mathcal{M} \models \phi$ for every sentence $\phi \in KB$, then $\mathcal{M} \models \alpha$.

In other words, there exists no logically possible world that satisfies the knowledge base while simultaneously falsifying the query.

This definition makes clear that entailment is a purely *semantic* notion. It depends on the meaning of the sentences in the knowledge base and the set of models they admit, rather than on any specific algorithm used to perform inference. Two different inference procedures may reach the same conclusion by different reasoning paths, yet both are correct as long as the semantic condition of entailment is satisfied.

Entailment versus derivation: It is important to distinguish entailment from *derivability*. Derivability, written as:

$$KB \vdash \alpha,$$

refers to the existence of a formal syntactic proof of α from *KB* within a specified proof system. While entailment concerns truth across all models, derivation concerns what can be proven using formal symbol manipulation.

An inference system is said to be *sound* if every sentence it can derive is also semantically entailed, and *complete* if every entailed sentence can, in principle, be derived. These properties ensure that syntactic reasoning aligns correctly with semantic meaning.

Intuitive Meaning of Entailment

Intuitively, stating that $KB \models \alpha$ means that the information contained in the knowledge base is sufficient to *guarantee* the truth of α. There is no need to introduce additional assumptions, guesses or probabilistic reasoning. In every situation consistent with what the knowledge base asserts, the query must hold.

If even a single logically possible situation exists in which all statements in the knowledge base are true while α is false, then the query is *not* entailed. In such a case, the knowledge base does not justify accepting α as a logically valid conclusion.

Role of queries in intelligent systems: Queries serve as the primary interface between an intelligent system and its environment or users. By evaluating entailment, the system can answer questions, verify hypotheses, detect inconsistencies, and support decision-making. In classical AI, the explicit separation between knowledge storage and query answering is essential, as it allows the same knowledge base to be used flexibly for many different reasoning tasks.

Understanding queries and entailment thus provides the semantic foundation for all subsequent inference techniques, including forward and backward chaining, resolution, and planning as logical reasoning, which are examined in the following sections:

Illustrative example (Multistep Entailment)

Consider a slightly richer knowledge base drawn from a simple organizational domain. Suppose the knowledge base contains the following sentences:

$$Manager(Alice, Bob),$$
$$Manager(Bob, Carol),$$
$$\forall x \forall y\ (Manager(x, y) \rightarrow Employee(y)),$$
$$\forall x \forall y \forall z\ (Manager(x, y) \wedge Manager(y, z)) \rightarrow IndirectManager(x, z)\ .$$

The first two sentences are *ground facts* stating that Alice manages Bob and Bob manages Carol. The third sentence is a general rule asserting that anyone who is managed by someone is an employee. The fourth sentence defines an *indirect managerial relationship*: If x manages y and y manages z, then x is an indirect manager of z.

Now consider the following query:

$$IndirectManager(Alice, Carol).$$

This query is entailed by the knowledge base. In every model in which all four sentences are true, the managerial relationships must satisfy the stated rule. Since Alice manages Bob and Bob manages Carol, the conditions of the rule are met, and the conclusion necessarily follows. There exists no interpretation consistent with the knowledge base in which the query could be false. Therefore,

$$KB \models IndirectManager(Alice, Carol).$$

Derived entailment through intermediate facts: The same knowledge base also entails the query:

$$Employee(Carol).$$

Although this fact is not stated explicitly, it follows logically from the third rule. Because Bob manages Carol, the rule $Manager\ (x, y) \rightarrow Employee\ (y)$ applies, guaranteeing that Carol is an employee in every model of the knowledge base. This illustrates how entailment allows an intelligent system to derive implicit knowledge from explicitly stored facts and rules.

Nonentailment and insufficient information: Now consider the query:

$$Manager(Alice, Carol).$$

This query is *not* entailed by the knowledge base. While Alice indirectly manages Carol, there is no rule asserting that indirect management implies direct management. Consequently, there exist models in which Alice manages Bob, Bob manages Carol, and yet Alice does not directly manage Carol. Since such a counterexample is logically possible, the query is not entailed.

Failure of entailment versus falsehood: It is important to emphasize that the lack of entailment does not imply that the query is false. The knowledge base simply does not contain enough information to guarantee the truth of the query. Additional rules—such as a definition equating indirect and direct management—would be required to establish entailment. This distinction highlights the conservative nature of logical reasoning—conclusions are accepted only when they are logically justified by the available knowledge.

Role of Inference Mechanisms

In practice, entailment in such examples is determined by inference mechanisms such as forward chaining, backward chaining or resolution. A forward chaining system would derive new facts like *Employee*(*Carol*) and *Indirect Manager* (*Alice, Carol*) by repeatedly applying rules to known facts. A backward chaining system would instead attempt to prove the query by working backward from the goal and checking whether the necessary conditions can be satisfied from the knowledge base.

These inference procedures are *sound* if every conclusion they produce is semantically entailed by the knowledge base, and *complete* if they can derive every sentence that is entailed. Together, they provide the operational machinery that allows abstract logical entailment to be realized as concrete reasoning within an intelligent system.

10.3.5 Consistency and Monotonicity

Two fundamental logical properties govern the behavior of classical knowledge-based systems: *Consistency* and *monotonicity*. Together, these properties define the logical discipline under which knowledge is represented, queried, and extended in classical AI. They ensure that reasoning is mathematically well-founded, predictable, and free from ambiguity, albeit at the cost of reduced flexibility in dynamic or uncertain environments.

Consistency

A knowledge base is said to be *consistent* if it does not contain any logical contradictions. Formally, consistency requires that there exists at least one interpretation (or model) in which all sentences in the knowledge base are simultaneously true. If no such interpretation exists, the knowledge base is inconsistent.

Equivalently, a knowledge base *KB* is consistent if there is no sentence α such that:

$$KB \models \alpha \quad \text{and} \quad KB \models \neg\alpha.$$

Consistency is a minimal requirement for meaningful reasoning. Without consistency, logical consequence becomes trivial, since every sentence becomes entailed.

Example of a consistent knowledge base: Consider the following simple knowledge base drawn from a familiar domain:

$$Human(Vyāsa),$$
$$\forall x\,(Human(x) \rightarrow Mortal(x)).$$

The first sentence asserts a concrete fact: *Vyāsa* is a human. The second sentence expresses a general law of the domain, stating that all humans are mortal. These two sentences are logically compatible and can be true at the same time.

Formally, the knowledge base is *consistent* because there exists at least one interpretation (model) in which both sentences hold simultaneously. In such a model, *Vyāsa* belongs to the set of humans, and the implication from being human to being mortal is satisfied.

Under these conditions, the inference mechanism can legitimately derive the conclusion:

$$Mortal(Vyāsa),$$

as a direct logical consequence of the knowledge base. This conclusion follows without ambiguity and does not introduce any contradiction, illustrating how consistent knowledge supports reliable and meaningful reasoning.

Inconsistency and logical explosion: Now suppose that the knowledge base is extended by adding both of the following sentences:

$$Mortal(Vyāsa) \quad \text{and} \quad \neg Mortal(Vyāsa).$$

The knowledge base has now become *inconsistent*, because no single interpretation can satisfy both a statement and its negation at the same time. In other words, there is no possible model in which *Vyāsa* is both mortal and not mortal simultaneously.

In classical logic, inconsistency has particularly severe consequences due to the principle known as *logical explosion* (*ex contradictione quodlibet*). This principle states that once a contradiction is present, *any* sentence whatsoever can be derived from it. Formally, if there exists a sentence α such that:

$$KB \models \alpha \quad \text{and} \quad KB \models \neg\alpha,$$

then for any sentence γ:

$$KB \models \gamma.$$

As a consequence, an inconsistent knowledge base loses its ability to support meaningful inference. Since every conclusion becomes derivable, the system can no longer distinguish between valid, informative conclusions and arbitrary or nonsensical ones. Preventing inconsistency is therefore a central requirement in the design, verification, and maintenance of classical knowledge-based systems, ensuring that logical reasoning remains sound and trustworthy.

Monotonicity

In addition to consistency, classical logic-based knowledge bases obey the principle of *monotonicity*. Monotonic reasoning means that the set of entailed conclusions can only *increase* as new knowledge is added; previously derived conclusions are never invalidated.

Formally, classical logical consequence is monotonic:

$$\text{If } KB \subseteq KB', \text{ and}$$
$$KB \models \alpha,$$

then

$$KB' \models \alpha.$$

That is, adding new information cannot cause a previously entailed sentence to cease being entailed.

Illustration of monotonic reasoning: Suppose a knowledge base has already been shown to entail the sentence:

$$Mortal(Vyāsa).$$

This conclusion follows logically from the information currently stored in the knowledge base and is therefore guaranteed to be true in every model that satisfies that knowledge.

Now consider extending the knowledge base by adding the following new facts:

$$Author(Vyāsa), \qquad Composed(Vyāsa, Mahābhārata).$$

These additional sentences introduce new information about *Vyāsa*, but they do not contradict any existing statement. As a result, all models that satisfied the original knowledge base can be extended to satisfy the enlarged knowledge base as well. Consequently, the conclusion:

$$Mortal(Vyāsa)$$

remains true after the extension.

This example illustrates the *monotonic* nature of classical logical reasoning: Adding new knowledge can increase the set of derivable conclusions, but it cannot invalidate conclusions that were previously entailed. While the newly added facts may enable further inferences—such as conclusions about authorship or creative activity—they do not undermine earlier deductions.

Monotonicity allows reasoning to proceed incrementally and cumulatively. Once a conclusion has been established, it can be safely reused in subsequent reasoning without the need to retract or reevaluate it, greatly simplifying the design and correctness of inference mechanisms in classical AI.

Implications for inference in AI systems: Monotonicity greatly simplifies the design and analysis of inference mechanisms. Because conclusions never need to be retracted, inference procedures can be proven sound and complete with respect to the underlying logic. Reasoning becomes cumulative, and proofs remain valid as the knowledge base grows.

At the same time, monotonicity introduces important limitations. Many real-world reasoning tasks require revising beliefs in light of new information. Discovering an exception, correcting an erroneous assumption or handling incomplete knowledge may require withdrawing earlier conclusions. Such belief revision is not naturally supported within monotonic logical systems.

Classical Perspective

Despite these limitations, consistency and monotonicity are defining features of classical knowledge representation and reasoning. They provide a clean, precise, and mathematically rigorous foundation for AI, particularly in domains where knowledge is stable, explicit, and well-understood.

More expressive frameworks—such as nonmonotonic logics, default logic, and probabilistic reasoning—have been developed precisely to relax these assumptions. However, classical AI deliberately adopts consistency and monotonicity as foundational principles, ensuring that reasoning remains logically sound, predictable, and amenable to formal analysis.

10.3.6 Role of Knowledge Bases in Intelligent Systems

Knowledge bases form the intellectual core of many classical AI systems. They provide a structured and explicit representation of domain knowledge that enables systems to reason, explain, and act in a principled manner. Rather than responding purely through preprogrammed reactions, knowledge-based systems use stored facts and rules to guide their behavior toward well-defined goals.

The role of knowledge bases is particularly evident in the following classes of intelligent systems:

- **Expert systems:** In expert systems, the knowledge base encodes the expertise of human specialists in a particular domain, such as medicine, engineering or fault diagnosis. The inference engine applies logical reasoning to this knowledge to provide recommendations, diagnoses or decisions. The explicit structure of the knowledge base allows such systems to explain their conclusions by tracing them back to underlying facts and rules.
- **Planning systems:** Planning systems rely on knowledge bases to represent actions, preconditions, effects, and goals. Logical reasoning over this knowledge enables the system to generate sequences of actions that transform an initial state into a desired goal state. Here, the knowledge base serves as a model of how the world changes in response to actions.
- **Automated theorem provers and logic-based agents:** In automated reasoning systems, the knowledge base contains axioms and theorems expressed in formal logic. Inference mechanisms attempt to derive new theorems or verify the truth of conjectures. Logic-based agents similarly use knowledge bases to reason about their environment, their own actions, and the actions of other agents.
- **Natural language understanding systems:** Symbolic natural language systems use knowledge bases to represent the meanings of words, sentences, and discourse structures. By reasoning over this symbolic knowledge, such systems can interpret user input, resolve ambiguities, and draw implicit conclusions that are not explicitly stated in the text.

Across these applications, knowledge bases enable AI systems to operate at a conceptual level. They allow systems to reason about *what is true*, *why it is true*, and *what follows from it*. This capability moves AI beyond purely reactive or pattern-driven behavior toward deliberate, explainable, and goal-directed intelligence.

From a classical AI perspective, the presence of an explicit knowledge base is what distinguishes a system that merely computes from one that *reasons*. It provides the foundation upon which inference, planning, explanation, and intelligent decision-making are built, making knowledge bases a central and enduring component of symbolic AI.

10.4 KNOWLEDGE CYCLE IN INTELLIGENT SYSTEMS

An intelligent system is not merely a repository of stored facts, nor simply a collection of isolated algorithms. Its defining capability lies in the continuous and structured interaction between knowledge, reasoning, and action. This interaction can be understood as a recurring *knowledge cycle*, illustrated in Figure 10.3.

As shown in the figure, **knowledge representation** occupies the central position within the system. Around it operates a cyclic process consisting of Perception, Representation, Knowledge Base, Inference, Planning, Action, and Update. These components do not function independently; rather, they form a coordinated loop that enables adaptive and goal-directed intelligence.

The cyclic structure captures the dynamic nature of intelligent behavior. Knowledge evolves as the system interacts with its environment. Each iteration refines the internal model, supports more informed reasoning, and enables more effective decision-making. The stages of the knowledge cycle are:

1. **Perception:** The cycle begins with *perception*. The system acquires information from its environment through sensors, user input, databases or communication with other agents. At this stage, the information exists as raw data. It may be incomplete, noisy or partially observable. Perception provides the external input that initiates the cycle.
2. **Representation:** Perceived data must be translated into structured symbolic form. This stage converts raw input into internal representations consistent with the system's ontology and representation language.

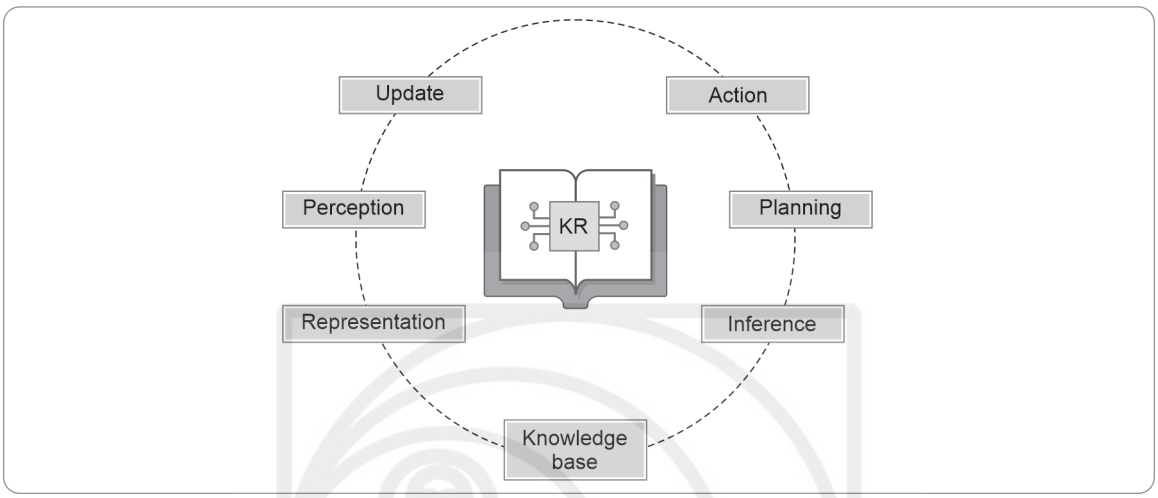

Fig. 10.3: Knowledge cycle in intelligent systems, with knowledge representation at the core

Facts are asserted, objects instantiated, and relations established. Through representation, information becomes meaningful knowledge.

3. **Knowledge base:** The structured information is stored in the *knowledge base*, which serves as the system's internal model of the world. Formally, the knowledge base can be viewed as a set of logical sentences:

$$KB = \{\phi_1, \phi_2, \ldots, \phi_n\}.$$

These sentences encode facts, rules, and constraints. The knowledge base itself does not perform reasoning; it provides the content upon which inference operates. In Figure 10.3, this component forms the foundational layer supporting reasoning and planning.

4. **Inference:** The inference mechanism operates over the knowledge base to derive new conclusions and answer queries. It determines whether a statement α is logically entailed:

$$KB \models \alpha.$$

Inference produces implicit knowledge from explicit representations, thereby expanding the system's understanding without additional input.

5. **Planning:** Given goals and current knowledge, the system reasons about possible courses of action. Using action models defined by preconditions and effects, it constructs sequences:

$$\langle a_1, a_2, \ldots, a_k \rangle$$

that transform the current state into one satisfying the goal. Planning extends logical reasoning into the temporal dimension, allowing anticipation of future states.

6. **Action:** The selected plan is executed. Actions interact with the environment and produce state transitions:

$$s \xrightarrow{a} s'$$

Execution completes the outward phase of the cycle, altering both the environment and the system's internal assumptions.

7. **Update:** After action, new perceptions are obtained. The system compares expected and observed outcomes, incorporates new facts, and checks consistency. The knowledge base is updated accordingly. This closes the loop and prepares the system for the next cycle.

Conceptual significance: The knowledge cycle unifies the components of classical AI into a coherent architecture. Declarative knowledge provides content, inference derives consequences, planning guides action, and execution generates new information that feeds back into the system. This perspective emphasizes that intelligence emerges not from any single component in isolation, but from the disciplined coordination of representation, reasoning, and action within a structured and recurring process. In this sense, the knowledge cycle provides the architectural foundation for knowledge-based intelligent systems.

10.5 REPRESENTING ACTIONS AND CHANGE

While earlier sections focused on representing *static knowledge* about the world—facts, properties, and relationships that hold at a particular instant—intelligent agents must also reason about *dynamics*: How the world evolves over time as a consequence of actions. An AI system that cannot represent change is confined to passive interpretation of its environment. To behave intelligently, it must be able to anticipate the outcomes of its actions, compare alternative courses of action, and select those that lead toward desired goals.

Reasoning about actions introduces a temporal dimension into knowledge representation. Instead of asking only *what is true now*, an intelligent agent must also reason about *what will be true after an action is performed* and *what remains unchanged*. For example, a robot navigating a building must understand that moving to a new room changes its location, but does not alter the layout of the building or the identities of objects within it.

In classical AI, the problem of representing actions and change is addressed by explicitly modeling how actions relate one situation or state of the world to another. Actions are treated as well-defined operators that transform states in systematic and predictable ways. Such representations allow the agent to simulate future states, reason about sequences of actions, and evaluate whether a particular plan will achieve a given goal.

This explicit treatment of actions and their effects provides the semantic foundation for key AI capabilities such as automated planning, goal-directed problem-solving, and deliberate decision-making. By encoding how the world changes in response to actions, knowledge representation moves beyond static description and becomes a tool for understanding, predicting, and controlling dynamic environments.

10.5.1 Actions as State Transformations

In classical AI, an *action* is formally understood as an operator that transforms one *state* of the world into another. A state is typically defined as a complete assignment of truth values to the relevant fluents of the domain at a particular moment. Acting, therefore, corresponds to moving from one state description to a new one by changing the truth values of certain propositions.

To represent an action precisely, classical AI systems decompose it into two fundamental components:

- **Preconditions:** Logical conditions that must hold in the current state for the action to be applicable. If the preconditions are not satisfied, the action cannot be executed in that state.
- **Effects:** Logical descriptions of how the execution of the action modifies the state of the world. Effects specify which facts become true and which facts become false after the action is performed.

This precondition–effect structure allows an intelligent agent to reason about actions symbolically. By checking preconditions, the agent can determine whether an action is feasible in the current situation. By applying the effects, the agent can predict the resulting state and evaluate whether it brings the system closer to a desired goal.

Example 1. A Robot Movement Action

Consider a simple robotic action *Move*(*Robot, A, B*), which moves a robot from location *A* to location *B*. Such movement actions are fundamental in navigation and planning problems, where an agent must reason about how its position changes over time in response to its own actions.

A natural logical representation of this action is:

- **Preconditions:**

$$At(Robot, A)$$

This expresses that the robot must currently be at location *A* in order for the move to be possible. If the robot is not at *A*, the action is not applicable and should not be considered by the planning or control system.

- **Effects:**

$$\neg At(Robot, A), \qquad At(Robot, B)$$

These effects state that after the action is executed, the robot is no longer at *A* and is now at *B*. The effects explicitly capture the minimal change required to reflect the outcome of the action.

This representation makes explicit both the *requirements* for executing the action and the *changes* it induces in the world state. Only the robot's location is modified; all other facts not mentioned in the effects—such as the positions of other objects, the connectivity of locations or the identity of the robot—are assumed to remain unchanged. This assumption of persistence is central to classical action representations and enables concise modeling of dynamic systems.

State transition perspective: From a state-transition viewpoint, an action defines a directed relation between states:

$$s \xrightarrow{Move(Robot, A, B)} s'$$

where *s* is a state in which the preconditions hold and *s′* is the resulting state obtained by applying the action's effects. In state *s*, the robot occupies location *A*; in state *s′*, that fact has been replaced by the assertion that the robot occupies location *B*.

This perspective allows sequences of actions to be composed naturally. For example, a navigation plan can be viewed as a path:

$$s_0 \xrightarrow{Move(Robot, A, B)} s_1 \quad s_1 \xrightarrow{Move(Robot, B, C)} s_2$$

where each action transforms the current state into a successor state.

Example 2. Smart Home Energy Control Action

Consider an intelligent smart-home system responsible for managing energy consumption. One of its tasks is to control household appliances in order to reduce power usage during peak hours while maintaining user comfort. Actions in this domain modify the operational state of devices rather than physical locations.

Suppose the system includes the action:

$$TurnOff(AirConditioner).$$

A logical representation of this action can be specified as follows:

- **Preconditions:**

$$On(AirConditioner)$$

The action is applicable only if the air conditioner is currently running.

- **Effects:**

$$\neg On(AirConditioner), \qquad EnergySavingMode$$

After execution, the air conditioner is turned off, and the system enters an energy-saving mode.

This representation allows the AI system to reason symbolically about energy management. If the precondition does not hold—for example, if the air conditioner is already off—the action is not applicable and should not be selected.

Reasoning about change: Executing *TurnOff(AirConditioner)* modifies only a small portion of the world state. While the operational status of the air conditioner changes, other aspects of the environment remain unaffected:

$$Temperature(Room), \quad Occupancy(Room), \quad TimeOfDay$$

are assumed to persist unless altered by other actions or external events.

This highlights a key aspect of action representation: Actions specify *what changes*, not *everything that remains the same*. Such selective change modeling enables efficient reasoning without explicitly reasserting all unaffected facts.

State transition interpretation: From a state-based perspective, the action induces a transition:

$$s \xrightarrow{TurnOff(AirConditioner)} s'$$

where:

- s satisfies $On(AirConditioner)$.
- s' satisfies $\neg On(AirConditioner)$ and $EnergySavingMode$.

This transition can be composed with other actions, such as *LowerBlinds(Room) or DelayDishwasher()*, allowing the system to reason about multistep energy-optimization plans.

10.5.2 Reasoning about Sequences of Actions

Once individual actions are represented as state-transforming operators, an intelligent agent can reason not only about isolated actions, but also about *sequences of actions* and their cumulative effects. This capability is essential for goal-directed behavior, since most nontrivial tasks cannot be accomplished by a single action but require a coordinated series of steps executed in a particular order.

Formally, a sequence of actions

$$\langle a_1, a_2, \ldots, a_n \rangle$$

induces a sequence of state transitions

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} s_n$$

where s_0 is the initial state of the world and each subsequent state s_{i+1} results from applying the effects of action a_i to state s_i, provided that the preconditions of a_i are satisfied in s_i. Reasoning about action sequences therefore requires ensuring that *every action in the sequence is applicable at the point where it is executed*.

Given an initial state and a desired goal state, an AI system can view planning as the problem of discovering a sequence of actions whose combined effects transform the initial state into one that satisfies the goal. Rather than reasoning about actions in isolation, the system reasons about how the effects of earlier actions enable the preconditions of later ones. This interdependence among actions is a defining feature of planning problems.

Illustrative example: Consider a robot operating in an environment with locations *A*, *B*, and *C*. Suppose the initial state contains the fact:

$$At(Robot, A),$$

and the goal is:

$$At(Robot, C).$$

Assume the system knows the action schemas:

$$Move(Robot, A, B), \qquad Move(Robot, B, C).$$

By reasoning about the effects of these actions, the system can infer that executing *Move(Robot, A, B)* will produce a state in which:

$$At(Robot, B)$$

holds. In that resulting state, the preconditions for *Move(Robot, B, C)* are satisfied, allowing the second action to be executed. The combined effect of the action sequence

$$\langle Move(Robot, A, B),\ Move(Robot, B, C)\rangle$$

is therefore a state in which the goal condition $At(Robot,C)$ holds.

Planning as logical reasoning: From this perspective, planning can be understood as a form of logical reasoning about actions and states. The agent reasons forward from the initial state by applying actions and their effects or backward from the goal by reasoning about which actions could have produced it. In both cases, the underlying representation of actions as precondition–effect rules enables systematic prediction of future states.

10.5.3 The Frame Problem

The explicit representation of actions and their effects gives rise to a fundamental and historically important difficulty in knowledge representation known as the *frame problem*. At its core, the frame problem concerns how to represent change in a dynamic world *without* having to explicitly state everything that does *not* change when an action is performed.

In most real-world situations, an action affects only a small subset of the properties of the world. The vast majority of facts remain unchanged. For example, when a robot executes a movement action from one room to another, its location changes, but its identity, color, physical structure, ownership, and many other properties typically remain the same. Similarly, moving a book from one shelf to another changes the book's location, but not its title, author or number of pages.

If every action were represented by explicitly listing both its effects and all of its *noneffects*, the resulting knowledge base would be enormous and unmanageable. For each action, one would need to assert that every unaffected fluent remains unchanged. Such a representation would scale poorly as the number of actions and fluents increases.

Formal Statement of the Problem

Formally, the frame problem can be stated as follows:
How can an intelligent agent represent the effects of actions while assuming, by default, that all other aspects of the world remain the same, without explicitly encoding those noneffects?

This question highlights a fundamental tension in knowledge representation—on the one hand, the representation must be expressive enough to capture the relevant effects of actions; on the other hand, it must be compact and efficient enough to support reasoning in complex domains.

Illustrative example: Consider again the action *Move(Robot, A, B)* with effects:

$$\neg At(Robot, A), \qquad At(Robot, B).$$

In a naive representation, one might also need to specify statements such as:

$$Color(Robot) = Red, \qquad BatteryLevel(Robot) = High, \qquad ID(Robot) = R1,$$

and assert that each of these remains unchanged after the move. Repeating this for every action and every property quickly becomes infeasible.

10.5.4 Classical Approaches and Limitations

To reason formally about actions and change, classical AI developed a number of logical frameworks that make the effects of actions explicit while attempting to control representational complexity. Among the most influential of these are the *situation calculus*, the *event calculus*, and *STRIPS-style* action representations. Each provides a principled way to describe how actions transform the world, while also highlighting the inherent trade-offs between expressiveness, clarity, and computational efficiency.

Situation Calculus

The situation calculus is a first-order logical formalism in which the world is described in terms of *situations*, representing snapshots of the world at different points in time. Actions are modeled as functions that map one situation to another, and *fluents* are predicates whose truth values may vary across situations. For example, a fluent such as *At*(*Robot, Location, s*) states that the robot is at a given location in situation *s*.

By explicitly indexing fluents with situations, the situation calculus provides a clear and rigorous semantics for reasoning about action sequences. However, it traditionally requires the use of numerous frame axioms to specify which fluents remain unchanged, making naive formulations verbose and difficult to scale.

Event Calculus

The event calculus takes a different perspective by focusing on *events* and their effects over time. Rather than reasoning directly about successive situations, it models how events initiate or terminate properties, and how these properties persist unless affected by subsequent events. This approach aligns more naturally with continuous or overlapping events and temporal reasoning.

While the event calculus offers greater flexibility for representing time and concurrency, it can become complex to implement and reason with, particularly in large-scale or real-time domains. Inference in rich temporal models may also be computationally expensive.

STRIPS-Style Representations

STanford Research Institute Problem Solver (STRIPS)-style representations[1], originally developed for automated planning systems, describe actions using three components: In STRIPS-style planning, an action is represented as a triple ⟨Pre, Add, Del⟩ consisting of preconditions, an add list, and a delete list. An action specifies which facts must be true before execution, which facts become true afterward, and which facts are no longer true. This representation is simple, compact, and well-suited for planning algorithms. Figure 10.4 presents the STRIPS-style representation of an action. By assuming that all facts not mentioned in the add or delete lists remain unchanged, STRIPS implicitly addresses the frame problem. However, this simplicity comes at the cost of expressive power. STRIPS-style representations struggle to model conditional effects, uncertainty, continuous change, and complex interactions among actions.

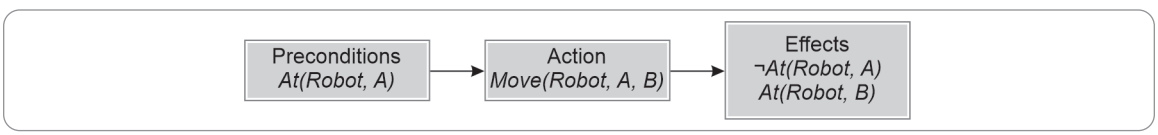

Fig. 10.4: STanford Research Institute Problem Solver-style representation of an action as a precondition-effect structure

[1]https://ai.stanford.edu/nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf

Limitations of Classical Logical Approaches

Despite their elegance and conceptual clarity, classical logical approaches to action and change face important limitations. They typically assume complete and accurate knowledge of the world, deterministic actions, and instantaneous state transitions. In many real-world environments, agents must operate under uncertainty, partial observability, and time constraints. Moreover, purely logical representations can be brittle when confronted with noisy sensor data or unpredictable external influences. As a result, reasoning and planning systems based solely on classical logic may fail to scale or adapt effectively in dynamic, real-time settings. These limitations motivate the development of more advanced planning and reasoning frameworks, including probabilistic planning, decision-theoretic models, and learning-based approaches. Nevertheless, classical formalisms remain foundational, providing the conceptual and methodological groundwork upon which modern AI systems are built.

10.6 CLASSICAL KNOWLEDGE REPRESENTATION SCHEMES

The abstract principles of knowledge representation discussed so far—declarative structure, inference, consistency, and action modeling—must ultimately be realized through concrete representational schemes. Classical AI developed a number of such schemes to encode knowledge about the world in explicit, structured, and interpretable forms. Each scheme embodies a particular view of what knowledge is important, how it should be organized, and how reasoning should proceed over it.

These classical knowledge representation schemes do not compete with one another so much as emphasize different aspects of intelligent reasoning. Some focus on logical precision, others on conceptual organization, and still others on capturing common-sense structure or linguistic meaning. What unifies them is their symbolic nature—knowledge is represented explicitly using discrete symbols, relations, and structures that stand for entities and phenomena in the world.

10.6.1 Ontological Commitments and Conceptualization

Every knowledge representation embodies an *ontological commitment*: A decision about what kinds of entities are assumed to exist in the modeled world. Before reasoning or planning, an AI system must adopt a *conceptualization*—a structured view specifying which entities, distinctions, and relations are represented.

Ontological commitments are modeling choices rather than metaphysical claims. Different systems may conceptualize the same domain differently depending on their tasks. For example, a navigation system may treat rooms and corridors as primitive entities, whereas a social reasoning system may focus on people, roles, and relationships. Both operate in the same environment but adopt different ontologies because their reasoning objectives differ.

Entities and Distinctions

A central question is: *What entities should be represented explicitly?* Classical AI commonly distinguishes among:

- **Objects:** Persistent entities such as *Robot*, *Book* or *Person*, typically assumed to retain identity over time.
- **Events and actions:** Occurrences such as *Move*, *PickUp* or *TurnOff* that bring about change.
- **Properties and relations:** Attributes and relationships, such as *Color(Robot)* or *At(Robot, Room)*.

Distinguishing these categories supports reasoning about persistence, change, causality, and interaction.

Objects versus Events: Separating enduring objects from transient events is foundational. Objects persist; events modify their properties. For example, a robot moving between rooms remains the same robot, though

its location changes. Without this distinction, reasoning about identity and temporal change becomes problematic.

Categories and abstraction: Ontological commitment also involves selecting appropriate categories (e.g., *Student, Vehicle, Appliance*). Categories enable abstraction—general knowledge can be stated once and applied to all members. They are modeling constructs chosen for representational utility. Overly fine distinctions increase complexity; overly coarse ones obscure relevant differences.

Conceptualization as a modeling choice: Together, these commitments define the system's conceptual vocabulary of objects, events, properties, and relations. They constrain subsequent representation and inference. In classical AI, conceptualization is evaluated pragmatically—a representation is appropriate insofar as it supports effective reasoning and explanation within its intended task domain. Different conceptualizations may therefore coexist for the same world.

10.6.2 Rule-Based Knowledge Representation

Rule-based knowledge representation encodes domain expertise as explicit *rules* describing how conclusions or actions follow from conditions. Instead of embedding knowledge in control flow, rule-based systems represent it declaratively as condition-action rules, making reasoning transparent and modular.

This mirrors human if-then reasoning: When certain conditions hold, specific consequences follow. Rule-based systems formalize this pattern for systematic machine execution.

IF-THEN rules: The basic unit is the *IF-THEN rule* (production rule), which states that whenever specified conditions are satisfied, certain conclusions may be inferred or actions taken. A rule has two components:

1. **Condition (antecedent):** Circumstances under which the rule applies
2. **Conclusion or action (consequent):** What is inferred or executed

Schematically:

IF condition THEN conclusion.

Conditions may combine predicates logically; conclusions may assert facts, update knowledge or trigger actions.

Example: Consider:

$$\text{IF } Sage(x) \wedge Composed(x, Text) \text{ THEN } Author(x).$$

This expresses that any individual who is a sage and has composed a text is an author. The variable x ensures general applicability. The rule states a relationship but does not specify how conditions are verified; that responsibility lies with the inference engine.

Rules are explicit, modular, and inspectable. Knowledge can be modified without altering the reasoning mechanism.

Rule base and working memory: Classical systems separate:

- **Rule base:** General IF–THEN rules encoding domain principles
- **Working memory:** Current factual assertions describing the specific situation

The rule base is relatively stable; working memory is dynamic and evolves during reasoning.

Illustrative example: Given the rule:

$$\text{IF } Sage(x) \wedge Composed(x, Text) \text{ THEN } Author(x),$$

and working memory:

$$Sage(Vyāsa), \qquad Composed(Vyāsa, Mahābhārata),$$

the inference engine matches the rule's conditions and derives:

$$Author(Vyāsa).$$

This inferred fact is added to working memory. New conclusions may trigger further rules, producing chains of inference.

Inference engine: The inference engine applies rules independently of domain content. Its operation involves:

- **Matching:** Identify rules whose conditions fit current facts.
- **Selection:** Choose among applicable rules.
- **Application:** Execute the chosen rule's conclusion.

This cycle incrementally extends working memory while preserving the separation between knowledge and control.

Forward chaining: Forward chaining is data-driven—reasoning begins with known facts and derives consequences.

Given:

$$Sage(Pāṇini), \quad Composed(Pāṇini, Aṣṭādhyāyī),$$

the system derives:

$$Author(Pāṇini).$$

With an additional rule:

$$\text{IF } Author(x) \text{ THEN } Scholar(x),$$

it further derives:

$$Scholar(Pāṇini).$$

Reasoning proceeds from available data toward broader conclusions.

Backward chaining: Backward chaining is goal-driven—reasoning begins with a query and works backward to establish supporting facts.

To determine whether:

$$Scholar(Patanjali)$$

holds, the system searches for a rule such as:

$$\text{IF } Author(x) \text{ THEN } Scholar(x),$$

and attempts to prove:

$$Author(Patanjali).$$

This may require verifying:

$$Sage(Patanjali), \quad Composed(Patanjali, YogaSutra).$$

If these are present, the goal is established.

Comparison of strategies: Both strategies use the same rule base and working memory but differ in direction: Forward chaining derives consequences; backward chaining tests hypotheses. Each is appropriate in different contexts.

Significance

Rule-based systems underpinned early expert systems and remain central to symbolic AI. Their explicit structure supports modularity, explanation, and principled reasoning, even though they face limitations in handling uncertainty and large-scale commonsense knowledge.

10.6.3 Semantic Networks

Semantic networks are an early and intuitive knowledge representation scheme in classical AI. They represent knowledge as a graph in which concepts and objects are connected by labeled relationships. Rather than using logical sentences, semantic networks emphasize *associative structure*—how entities are linked through meaningful relations.

This reflects the observation that human understanding is often organized relationally—who is related to whom, what belongs to what, and how one concept specializes another. Semantic networks encode such relationships directly.

Nodes as concepts and objects: Nodes represent concepts, categories or individual objects. A node may denote an abstract class such as *Sage* or *Text* or a specific instance such as *Vyāsa* or *Mahābhārata*. The choice of nodes reflects the adopted conceptualization.

For example:

Sage, Text, Vyāsa, Pāṇini, Mahābhārata, Aṣṭādhyāyī.

Some nodes denote general categories; others denote particular instances, allowing both general knowledge and specific facts to coexist in one structure.

Edges as relationships: Edges encode labeled relationships between nodes. Common relations include:

- **is-a** (or **instance-of**) for classification
- **part-of** for structural composition

For example:

$$Vyāsa \xrightarrow{\text{is-a}} Sage,$$

$$Mahābhārata \xrightarrow{\text{is-a}} Text,$$

$$Aṣṭādhyāyī \xrightarrow{\text{part-of}} GrammarTradition.$$

These links capture classification and structure without explicit logical formulation.

Inheritance of properties: A central feature of semantic networks is *inheritance*. If a node is connected *via* an *is-a* link to a more general node, it inherits that node's properties.

For example:

$$Sage \xrightarrow{\text{has-property}} Scholar,$$

$$Vyāsa \xrightarrow{\text{is-a}} Sage.$$

Thus *Vyāsa* inherits the property *Scholar*. Inheritance reduces redundancy and supports reasoning across levels of abstraction.

Figure 10.5 shows a network of interconnected concepts and instances. *is-a* links encode category membership, allowing inheritance, while associative edges (e.g., authorship) represent domain-specific relations. The graph structure visually conveys classification and association.

Illustrative example: Consider a network with:

- *Vyāsa* **is-a** *Sage,*
- *Pāṇini* **is-a** *Sage,*
- *Mahābhārata* **is-a** *Text,*
- *Aṣṭādhyāyī* **is-a** *Text,*
- *Vyāsa* **composed** *Mahābhārata,*
- *Pāṇini* **composed** *Aṣṭādhyāyī.*

The structure makes explicit both classification and authorship relations, allowing patterns to be recognized directly from the graph.

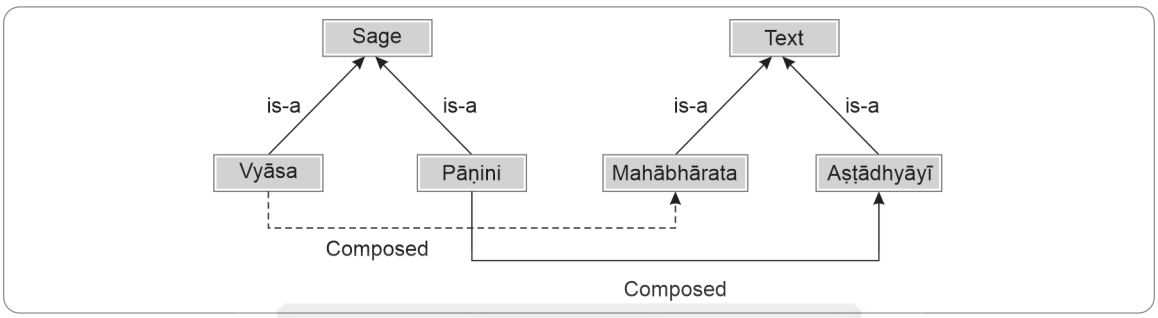

Fig. 10.5: A semantic network illustrating classification and associative relations. Nodes represent concepts and instances; labeled edges denote relationships such as *is-a* and *composed*

Strengths

- **Intuitive structure:** Mirrors relational human reasoning.
- **Visual clarity:** Easily represented as diagrams.
- **Inheritance support:** Avoids redundancy.
- **Extensibility:** Nodes and links can be added incrementally.

Limitations

- **Lack of formal semantics:** Interpretations may vary.
- **Ambiguity of relations:** Edge labels may lack precision.
- **Limited expressiveness:** Complex constraints and conditionals are difficult to represent.

10.6.4 Frames

Frames constitute an important classical knowledge representation scheme introduced to capture structured, object-centered knowledge in a form that is closer to how humans organize information about familiar situations and entities. Where semantic networks emphasize associative links among concepts, frames emphasize *internal structure*: What information is known about an entity, how that information is organized, and how typical values can be assumed in the absence of explicit data.

A frame can be understood as a data structure that represents a *stereotypical object, concept or situation*, together with its relevant attributes and expected properties. Frames were developed to address the need for representing rich, structured knowledge that goes beyond simple relational links, while remaining intuitive and interpretable.

Frames as Structured Objects

At a conceptual level, a frame represents a single entity or concept as a structured object. Each frame corresponds to a category (such as *Sage or Text*) or to a specific individual (such as *Vyāsa or Pāṇini*). Rather than representing knowledge as isolated facts, a frame groups related information together in a coherent unit.

For example, a frame representing the general concept *Sage* might include information about typical attributes associated with sages, such as scholarly activity, authorship, and teaching roles. This object-centered organization allows the system to reason about an entity as a whole rather than as a disconnected set of statements.

Slots and Fillers

The internal structure of a frame is defined by *slots* and *fillers*. A *slot* represents an attribute, role or property of the entity, while a *filler* specifies the value associated with that slot.

Conceptually, a frame may be viewed as a collection of attribute–value pairs. For instance, a frame for the individual sage *Vyāsa* may include the following slots and fillers:

- **Name:** *Vyāsa*
- **Role:** Sage
- **Composed:** *Mahābhārata*
- **Domain:** Itihasa

Each slot captures a distinct aspect of knowledge about the entity, and together they form a structured description that is easy to inspect, update, and extend. Slots may have single fillers, multiple fillers or even procedural attachments that specify how values should be computed or updated.

Default Values

One of the distinguishing features of frames is their support for *default values*. Defaults allow a frame to specify typical or expected values for slots that apply in most cases, without requiring those values to be stated explicitly for every instance.

For example, a general *Sage* frame might include default slots such as:

- **Language:** Sanskrit
- **Activity:** Teaching
- **Status:** Scholar

If a particular sage frame does not explicitly override these values, the system assumes the defaults apply. Thus, unless stated otherwise, the frame for *Pāṇini* would inherit the default assumption that his scholarly work is associated with Sanskrit.

Default values reflect common-sense reasoning—humans routinely assume typical properties unless there is evidence to the contrary. Frames encode this intuition directly, allowing knowledge representation to remain concise while still informative.

Inheritance in Frame Systems

Frames are often organized into hierarchies that support *inheritance*. A more specific frame may inherit slots and default values from a more general frame, while also introducing specialized information of its own.

For example, consider the following hierarchy:

Sage → *Grammarian*

The general *Sage* frame defines slots such as *Language* and *Activity*. A more specific *Grammarian* frame may inherit these slots and add new ones, such as:

- **Domain:** Grammar
- **PrimaryText:** *Aṣṭādhyāyī*

The individual frame for *Pāṇini* can then inherit information from both levels:

- **Language:** Sanskrit (inherited)
- **Activity:** Teaching (inherited)
- **Domain:** Grammar (specialized)
- **PrimaryText:** *Aṣṭādhyāyī* (instance-specific)

Inheritance reduces redundancy and ensures consistency by allowing shared knowledge to be defined once and reused across related entities.

Relation to Object-Oriented Thinking

Frames are closely related to ideas found in object-oriented programming, although they were developed independently within the AI community. Both frames and objects encapsulate data within structured entities, support inheritance, and promote modular design.

However, frames are primarily representational rather than computational. Their purpose is to model knowledge about the world, not to implement behavior through executable methods. This distinction reflects the classical AI emphasis on explicit knowledge representation and reasoning rather than on procedural execution.

The conceptual similarity between frames and objects highlights the influence of frame-based thinking on later software engineering paradigms, particularly in the design of systems that manage complex, structured information.

Frames versus the Frame Problem

It is essential to distinguish *frames* as a knowledge representation scheme from the *frame problem* discussed earlier in this chapter. Despite the similarity in terminology, the two concepts address entirely different issues.

Frames are a method for organizing knowledge about objects, concepts, and situations using structured slot–filler representations. They concern *what information is associated with an entity* and *how that information is organized*.

The frame problem, by contrast, concerns the difficulty of representing what does *not* change when actions are performed. It arises in reasoning about action and change, not in the internal structure of object representations.

Although both concepts share historical roots in classical AI, frames do not solve the frame problem directly. Rather, they address a different challenge—how to represent rich, common-sense knowledge in a compact and structured form.

Strengths and Limitations

Frames offer several advantages as a knowledge representation scheme. They are intuitive, structured, and well suited for representing typical knowledge about familiar entities and situations. Their support for defaults and inheritance aligns naturally with human expectations and common-sense reasoning.

At the same time, frames lack a universally agreed-upon formal semantics. Reasoning with frames often relies on informal conventions rather than rigorous inference rules. As a result, frame-based systems may struggle with ambiguity, exceptions, and complex interactions unless supplemented by additional mechanisms.

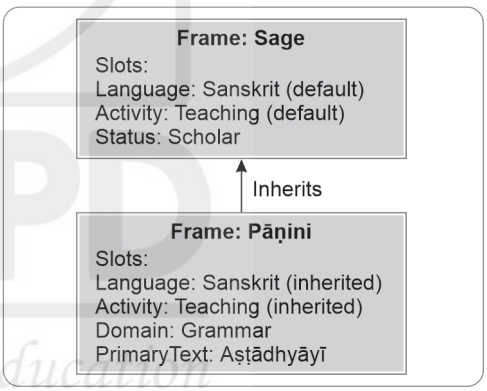

Fig. 10.6: Frame-based knowledge representation showing slots, default values, and inheritance. Frames were introduced as structured object-centered representations in classical artificial intelligence

Figure 10.6 illustrates the core idea of frame-based knowledge representation. Each frame corresponds to a structured object describing an entity, together with its associated slots and fillers. The upper frame represents the general concept *Sage*, which contains default slot values such as *Language* and *Activity*. The lower frame represents the individual *Pāṇini*, which inherits these default values while adding specialized information specific to his role as a grammarian.

This example highlights how frames organize knowledge locally within object-centered structures, support inheritance to avoid redundancy, and allow typical assumptions to be represented explicitly through default values.

Example 1. Frame for a Student

Consider a frame representing the general concept *Student*. This frame captures typical information associated with students as a structured object rather than as isolated facts.

- **Name:** Poulami
- **Role:** Student
- **Program:** Computer science
- **Status:** Enrolled (default)

If no additional information is provided, the system assumes that the student is actively enrolled. This illustrates how frames encode common-sense expectations through default slot values.

Example 2. Inheritance from a General Academic Frame

Suppose the knowledge base contains a more general frame *AcademicPerson*, which defines slots such as:

- **Affiliation:** University
- **Activity:** Learning or teaching

The *Student* frame inherits these slots automatically. As a result, the frame for *Poulami* inherits the assumption of university affiliation without explicitly restating it. Inheritance reduces redundancy and ensures consistency across related concepts.

Example 3. Default Values and Specialization

Consider a frame representing the general concept *Course*, with the following default slots:

- **Mode:** Classroom
- **Assessment:** Examination

Now consider a specialized frame *OnlineCourse*, which inherits from *Course* but overrides certain defaults:

- **Mode:** Online
- **Assessment:** Continuous evaluation

Frames allow such specialization by selectively overriding inherited defaults, reflecting how typical expectations change in specific contexts.

Example 4. Situation-Based Frame

Frames can also represent stereotypical situations rather than individual objects. For example, a *ClassroomLecture* frame may include:

- **Instructor:** Assigned teacher
- **Participants:** Students
- **Duration:** One hour (default)
- **Location:** Lecture hall

Such a frame captures structured knowledge about a recurring situation. Unless specified otherwise, the system assumes the default duration and location, illustrating how frames support efficient representation of routine events.

10.6.5 Conceptual Dependency Formalism

Conceptual Dependency (CD) formalism was developed in classical AI to represent the *meaning* of sentences independently of the specific words or syntactic structures used to express them. Its central premise is that different linguistic expressions may convey the same underlying conceptual event, and intelligent systems should reason about this shared meaning rather than about surface language.

For example, the sentences *'Vyāsa composed the Mahābhārata'* and *'The Mahābhārata was authored by Vyāsa'* differ grammatically yet express the same event. The CD represents both using a single, language-independent conceptual structure, enabling recognition of semantic equivalence across forms.

Language-independent meaning: CD abstracts away from grammar and encodes the *conceptual relationships* among agents, actions, and objects. Language is treated as a surface realization of deeper conceptual structures. Sentences with different phrasing—or even in different languages—map to the same representation if they describe the same situation. This abstraction supports paraphrase detection, inference, and meaning-level reasoning.

Primitive actions: Events in CD are expressed using a small set of *primitive actions* intended as universal conceptual building blocks (e.g., transfer, movement, creation, mental activity). Rather than enumerating thousands of verbs, CD decomposes complex actions into combinations of these primitives.

For example, the sentence *'Pāṇini composed the Aṣṭādhyāyī'*. is represented conceptually as a *creation* event:

- **Agent:** *Pāṇini*
- **Resulting object:** *Aṣṭādhyāyī*

Surface variations such as *'The Aṣṭādhyāyī. was written by Pāṇini'* map to the same conceptual structure.

The emphasis is not on the precise inventory of primitives, but on the principle that complex meanings can be constructed from a limited set of fundamental conceptual operations.

Role in early natural language processing (NLP): CD played a foundational role in early natural language understanding. By translating sentences into conceptual structures, systems could:

- Recognize paraphrases.
- Answer meaning-based questions.
- Connect linguistic input to structured world knowledge.

For instance, a system could answer *"Who authored the Mahābhārata?"* even if the text stated *"Vyāsa composed the Mahābhārata"*, because both share the same conceptual representation.

The CD thus functioned as a bridge between language and knowledge representation, enabling reasoning at the level of events and roles rather than words.

Historical significance and limitations: The CD marked a shift from syntactic to semantic representation in AI, emphasizing that understanding requires modeling concepts and events, not merely grammatical form.

However, defining a complete and universal set of primitive actions proved difficult, and scaling CD to large domains was challenging. Reasoning over rich conceptual structures often required additional mechanisms beyond representation alone.

Despite these limitations, CD significantly influenced later semantic models, knowledge graphs, and meaning-based representations. It remains an important milestone in the development of knowledge representation, reinforcing the goal of representing and reasoning about conceptual knowledge rather than manipulating surface symbols.

Illustrative Examples of Conceptual Dependency

To clarify how CD abstracts meaning away from linguistic form, consider the following pairs of sentences drawn from classical Indian scholarly contexts.

Example 1. Authorship as Conceptual Creation

Consider the two sentences:

1. "*Vyāsa* composed the *Mahābhārata*."
2. "The *Mahābhārata* was authored by *Vyāsa*."

Although the surface structure differs—one is active voice and the other passive—the underlying meaning is identical. In CD terms, both sentences describe a *creation event* in which:

- The *agent* is *Vyāsa*.
- The *resulting object* is the *Mahābhārata*.

The CD represents this shared meaning using a single conceptual structure, independent of grammatical voice or word choice. Thus, variations in syntax do not lead to different internal representations.

Example 2. Teaching as Knowledge Transfer

Consider the sentences:

"*Pāṇini* taught grammar to his students."

"Students learned grammar from *Pāṇini*."

From a linguistic perspective, one sentence emphasizes the teacher's action, while the other emphasizes the students' acquisition. CD abstracts both into a common conceptual event: The *transfer of knowledge*.

At the conceptual level:

- *Pāṇini* functions as the *source* of knowledge.
- Grammar is the *content* transferred.
- The students are the *recipients*.

By representing both sentences using the same underlying conceptual structure, CD allows an intelligent system to recognize them as semantically equivalent descriptions of a single event.

Example 3. Writing versus Composition

Consider the sentences:

"*Pāṇini* wrote the *Aṣṭādhyāyī*."

"The *Aṣṭādhyāyī*. came into existence through *Pāṇini*."

Despite their stylistic differences, both sentences describe the same conceptual occurrence—the creation of a scholarly text. The CD treats both as instances of a fundamental *create* or *produce* action, with *Pāṇini* as the agent and the *Aṣṭādhyāyī*. as the created entity.

This abstraction allows the system to reason about authorship, intellectual contribution or historical influence without relying on specific linguistic formulations.

Example 4. Narrative Understanding

The CD was particularly influential in early attempts to understand narratives. For example, consider the sentences:

"*Vyāsa* narrated the *Mahābhārata* to his disciples."

"The disciples heard the *Mahābhārata* from *Vyāsa*."

Here again, CD abstracts away from narrative perspective and focuses on the conceptual roles:

- *Vyāsa* as the source of narration
- The *Mahābhārata* as informational content
- The disciples as recipients

Such representations enabled early Natural Language Processing (NLP) systems to track who knew what, how knowledge spread, and how events unfolded across a story, even when the wording changed.

Example 5. Mathematical Problem-Solving

Consider the sentences:

"Poulami solved the equation."

"The solution of the equation was obtained by Poulami."

Although one sentence emphasizes the agent's action and the other emphasizes the outcome, both describe the same conceptual event. At the conceptual level, this corresponds to an *intellectual problem-solving action*, where:

- Poulami is the agent.
- The solution is the resulting abstract outcome.

The CD represents both sentences using a single conceptual structure, abstracting away from differences in voice and syntactic focus.

Example 6. Learning as Knowledge Acquisition

Consider the following pair:

"Aarohi learned calculus."

"Calculus was understood by Aarohi."

These sentences differ stylistically, yet convey the same meaning. The CD represents both as a *knowledge acquisition* event, in which:

- Aarohi is the recipient.
- Calculus is the acquired conceptual content.

By focusing on the underlying acquisition of knowledge, CD allows the system to treat both sentences as semantically equivalent.

Example 7. Teaching and Explanation

Consider the sentences:

"The teacher explained geometry to Poulami."

"Poulami received an explanation of geometry from the teacher."

One sentence highlights the act of explanation, while the other highlights the reception of information. At the conceptual level, both describe a *transfer of structured knowledge*, involving:

- The teacher as the source
- Geometry as the content
- Poulami as the recipient

The CD abstracts away from narrative perspective and represents the shared meaning of instructional interaction.

Example 8. Scholarly Writing and Documentation

Consider the following expressions:

"Aarohi documented the experimental results."

"The experimental results were recorded by Aarohi."

Despite their syntactic differences, both sentences describe the same conceptual event: The *recording of information*. The CD represents this as a single conceptual action in which Aarohi performs an intellectual activity resulting in the creation of a documented artifact.

STUDENT ASSIGNMENT

LONG ANSWER QUESTIONS

Answer the following questions with clear explanations, equations, and diagrams wherever appropriate.

1. Explain why knowledge representation is necessary beyond search-based techniques.
2. Describe the structure of a logical knowledge base with examples of facts, rules, and constraints.
3. Discuss consistency and logical explosion with suitable illustrations.
4. Explain the separation of knowledge and inference and its advantages.
5. Describe the knowledge cycle in intelligent systems with a neat diagram.
6. Compare forward chaining and backward chaining with examples.
7. Explain STRIPS-style action representation and discuss its limitations.
8. Describe semantic networks and illustrate inheritance with an example.
9. Explain frame-based knowledge representation with slots, fillers, and default values.
10. Discuss the conceptual dependency formalism and its role in natural language understanding.

SHORT ANSWER QUESTIONS

Answer each question briefly and precisely.

1. What is meant by explicit knowledge representation in classical AI?
2. State two differences between declarative and procedural knowledge.
3. Define a Knowledge Base (*KB*).
4. What is semantic entailment? Write its formal notation.
5. What is the role of an inference engine in a rule-based system?
6. Distinguish between facts and rules in a logical knowledge base.
7. What is monotonic reasoning?
8. Define preconditions and effects of an action.
9. What is inheritance in semantic networks?
10. What is the frame problem?

TRUE/FALSE QUESTIONS

State whether each statement is True or False.

1. A knowledge base stores knowledge in procedural form only.
2. If $KB \models \alpha$, then α is true in all models of *KB*.
3. In classical logic, inconsistency may lead to logical explosion.
4. Monotonic reasoning allows previously derived conclusions to be retracted.
5. Heuristic knowledge guarantees correctness of conclusions.
6. In STRIPS, facts not mentioned in add/delete lists are assumed to persist.
7. Frames primarily organize knowledge using slot–filler structures.
8. Semantic networks represent knowledge using graphs of nodes and edges.
9. Conceptual dependency focuses on syntactic structure rather than meaning.
10. In forward chaining, reasoning starts from known facts.

FILL IN THE BLANKS

Fill in the blanks with the most appropriate term or expression.

1. Knowledge representation enables systems to reason about _____ and relationships.
2. The notation $KB \models \alpha$ denotes logical _____.
3. A knowledge base is consistent if it contains no logical _____.
4. In rule-based systems, rules are stored in the _____.
5. In STRIPS, an action is represented using preconditions, add list, and _____.
6. The problem of representing what does not change after an action is called the _____ problem.
7. In semantic networks, category relationships are often represented using the _____ link.
8. Frames support typical assumptions through _____ values.
9. Backward chaining is a _____ -driven reasoning strategy.
10. In classical logic, adding new knowledge cannot invalidate previous conclusions due to _____.

MULTIPLE CHOICE QUESTIONS

Choose the most appropriate answer.

1. **The main purpose of knowledge representation is to:**
   a. Speed up hardware
   b. Store raw sensor data
   c. Encode structured domain knowledge for reasoning
   d. Replace inference engines
2. **A knowledge base is inconsistent if:**
   a. It contains many facts
   b. It has no rules
   c. It entails both α and $\neg\alpha$
   d. It supports inference
3. **Forward chaining is:**
   a. Goal-driven reasoning
   b. Data-driven reasoning
   c. Probabilistic reasoning
   d. Nonmonotonic reasoning
4. **In STRIPS representation, persistence of facts is handled by:**
   a. Explicit frame axioms
   b. Default add lists
   c. Implicit assumption that unaffected facts remain true
   d. Probabilistic inference
5. **Semantic networks primarily use:**
   a. Matrices
   b. Graph structures
   c. Differential equations
   d. Statistical models
6. **Frames organize knowledge using:**
   a. Slots and fillers
   b. Truth tables
   c. Resolution proofs
   d. Neural weights
7. **The frame problem concerns:**
   a. Slot inheritance
   b. Logical entailment
   c. Representing noneffects of actions
   d. Syntax parsing
8. **Conceptual dependency aims to represent:**
   a. Word frequency
   b. Surface grammar
   c. Deep conceptual meaning
   d. Statistical correlation
9. **Monotonic reasoning ensures that:**
   a. Conclusions may be withdrawn
   b. Adding knowledge reduces entailments
   c. Previously derived conclusions remain valid
   d. Inference becomes probabilistic
10. **Declarative knowledge specifies:**
    a. How to execute code
    b. What is true about the domain
    c. How to optimize search
    d. Hardware configuration

APPLICATION AND PROJECT-BASED QUESTIONS

These problems emphasize modeling, reasoning, and implementation.

1. Design a small logical knowledge base for a university domain including students, courses, and instructors.
2. Represent a medical diagnosis problem using rule-based knowledge representation.
3. Construct a semantic network for a library system with books, authors, and genres.
4. Develop a frame-based model for a smart classroom system.
5. Model a robotic navigation task using STRIPS-style actions.
6. Demonstrate forward and backward chaining on a given rule base.
7. Create a conceptual dependency representation for a short narrative paragraph.
8. Extend a knowledge base and analyze whether consistency is preserved.
9. Illustrate the frame problem using a real-world action example.
10. Compare two representation schemes for the same domain and evaluate their strengths and weaknesses.

ANSWER KEY

True/False Questions

1. False **2.** True **3.** True **4.** False **5.** False **6.** True **7.** True **8.** True **9.** False **10.** True

Fill in the Blanks

1. facts **2.** entailment **3.** contradictions **4.** rule base **5.** delete list **6.** frame **7.** is-a **8.** default **9.** goal **10.** monotonicity

Multiple Choice Questions

1. c **2.** c **3.** b **4.** c **5.** b **6.** a **7.** c **8.** c **9.** c **10.** b