QUESTION 1 – Vanilla meta-interpreter

The candidate is invited to introduce the "vanilla" meta-interpreter, to explain the meaning of each clause, and to show how it can be modified to support a right-most selection rule of the next subgoal from the resolvent.

ANSWER

The "vanilla" meta-interpreter is an interpreter for Prolog code written in Prolog code itself; for this reason, it is called a meta-interpreter.

% 1. Success: an empty goal (true) is solved.

solve(true) :- !.

% 2. Conjunction: to solve (A and B), solve A first and then B.

solve((A, B)) :- !, solve(A), solve(B).

% 3. Reduction: to solve A, find a clause (A :- B) and solve body B.

solve(A) :- clause(A, B), solve(B).

Explanation of the Clauses:

- Clause 1 (Unit Fact): Represents the success of the derivation. If the goal is true, the process terminates successfully. The "cut" (!) prevents the third clause from being invoked when the goal is specifically true.
- Clause 2 (Decomposition): Handles the comma (logical AND). It maintains the left-to-right order by solving A before moving on to B. The "cut" (!) ensures that a conjunction is not erroneously treated as a simple atom by the third clause.
- Clause 3 (Resolution): This is the core of the inference engine. The built-in predicate clause(A, B) searches the database for a rule whose head unifies with goal A and returns the body B. If A is a fact, B will simply be true.

2. Modification for the "Right-most Selection Rule"

To change the selection rule so the interpreter picks the right-most subgoal first, we swap the order of execution in the conjunction clause:

Prolog

% 2. Conjunction: to solve (A and B), solve B first and then A.

solve((A, B)) :- !, solve(B), solve(A).

Note: By switching the order to solve(B), solve(A), the interpreter will traverse the goal tree from right to left, effectively prioritizing the last subgoal in a conjunction.

QUESTION 2 – EC,Allen logic, LTL

The candidate is invited to briefly introduce the three paradigms for representing and reasoning over systems that evolve along the temporal dimension (namely, EC, Allen Interval Logic, and LTL).

(the point 1. Can be used also to answare this)

Answare

To represent and reason over systems that evolve over time, three primary paradigms are commonly used: Event Calculus (EC), Allen's Interval Logic, and Linear Temporal Logic (LTL). Each offers a different perspective on how time, events, and properties should be modeled.

1. **Event Calculus (EC)**

Proposed by Kowalski and Sergot, the Event Calculus is a logic-based framework that focuses on the happening of events and their effects on the world.

- Core Concepts: It reifies both fluents (properties that change over time) and events into first-order terms.
- Domain independent Axioms:
    - Happens(E, T): Event E occurs at time point T.
    - HoldsAt(F, T): Fluent F is true at time T.
    - Clipped(T1, F, T2): Fluent F became false between T1 and T2
- Domain Dependent Axioms (Reserved predicates):
    - Initiates(E, F, T): Event E cause F to hold at time T
    - Terminates(E, F, T): Event E cause F to cease at time T
    - Initialy(F): F is true at time 0
- Reasoning: EC naturally handles the Frame Problem by assuming that fluents persist (Hold) unless an event explicitly "clips" them (makes them false).

2. **Allen's Interval Logic**

Introduced by James Allen, this paradigm argues that intervals are more natural categories for human reasoning than discrete time points.

- Interval Relations: Instead of timestamps, it defines how two time intervals i and j relate to one another using 13 temporal operators.
- Basic Operators:
    - Interval: Begin(i), End(i)
    - Before(i, j): Interval i ends before j begins.
    - After(i,j): Before(j,i)
    - Meet(i, j): End(i) = Begin(j)
    - During(i, J): Begin(j)<Begin(i)<End(i)<End(j)
    - Overlap(i, j): Begin(i)<Begin(j)<End(i)<End(j)
    - Start(i,j): Begin(i) = Begin(j)
    - Finish(i,j): Finish(i) = Finish(j)
    - Equals(i,j): Begin(i) = Begin(j) AND Finish(i) = Finish(j)
- Use Case: It is highly effective for qualitative reasoning where the exact duration is unknown, but the relative order of events is critical.

3. **Linear Temporal Logic (LTL)**

LTL is a branch of Modal Logic where the accessibility relation between "possible worlds" is mapped to a discrete, linear sequence of time.

- Linear Time: It assumes that from any given moment, there is only one possible future path (a sequence of states).
- Temporal Operators: It introduces specialized modal operators to describe truth over time:
    - ◯ ϕ (Next): phi is true in the immediate next state.
    - □ ϕ (Always/Global): phi is true in all future states.
    - ◇ ϕ (Eventually): phi will be true at some point in the future.
    - ϕ U ψ (Until): phi holds until psi becomes true.
    - ϕ W ψ (Until Eventually): phi holds until psi eventually becomes true.
- Applications: LTL is widely used in Model Checking to verify system specifications, such as ensuring a "safety property" (something bad never happens) or a "liveness property" (something good eventually happens).

QUESTION 3 – OWA, CWA

The candidate is invited to briefly introduce the concepts of Open World Assumption (OWA) and Close World Assumption (CWA), and to illustrate their use in Prolog and in Description Logic.

Answare

Open World Assumption (OWA) vs. Closed World Assumption (CWA)

The Open World Assumption (OWA) and the Closed World Assumption (CWA) represent two fundamentally different ways of handling missing information in a knowledge base.

1. Concepts

- Closed World Assumption (CWA): Under this assumption, anything that is not explicitly stated as true in the knowledge base is assumed to be false. It assumes the knowledge base is complete.
- Open World Assumption (OWA): Under this assumption, if a sentence cannot be inferred from the knowledge base, its truth value is considered unknown rather than false. It acknowledges that the knowledge base may be incomplete

2. Use in Prolog (CWA)

Prolog operates under the Closed World Assumption, specifically through a mechanism called Negation as Failure (NAF)

In Prolog, the operator \+ (often read as "not") implements **Negation as Failure**. Unlike classical logic, where "not Q" means Q is provably false, in Prolog \+ Q means "Q cannot be proven true based on the current Knowledge Base."

p :- \+ q.

To evaluate p, Prolog attempts to prove q:

1. Sub-goal Execution: Prolog tries to satisfy q using the facts and rules in the Knowledge Base.
2. Finite Evaluation: If q is grounded (contains no uninstantiated variables), the search for a proof will terminate in finite time (assuming no infinite loops in the logic).
3. The Result: * If Prolog fails to find any proof for q (it finds "nothing in the KB that solves q"), it considers q to be false. Consequently, the negation \+ q succeeds, and p is evaluated as true.

3. Use in Description Logic (OWA)

Description Logics (DLs) are based on the Open World Assumption. This is particularly useful for the Semantic Web, where information is distributed and no single source can claim to have complete knowledge.

The absence of a fact does not mean a that fact does not exist; it simply means the information is missing.

The "Oedipus Example" of OWA Reasoning

KB Base:

1. **Iokaste** has a child named **Oedipus**.
2. **Oedipus** is a **Patricide** (he killed his father).
3. **Oedipus** and **Iokaste** has a child together named **Polyneikes**..
4. **Polyneikes** has a child named **Thersandros**.
5. **Thersandros** is **not a Patricide**

The Question: Does Iokaste have a child who is a **Patricide** AND who has a child that is **not a Patricide**?.

Normally we will not be able to respond to that question because we don't know anything about Polyneikes. But we can assume information about him.

Case 1: Polyneikes is a Patricide

If we assume Polyneikes is a patricide:

- He is a child of Iokaste.
- He is a **Patricide** (by our assumption in this case).
- He has a child (**Thersandros**) who is **not a Patricide**.
- **Conclusion:** In this world, the condition is **True** because of Polyneikes.

Case 2: Polyneikes is NOT a Patricide

If we assume Polyneikes is not a patricide:

- **Oedipus** is a child of Iokaste.
- Oedipus is a **Patricide**.

- Oedipus has a child (**Polyneikes**) who is **not a Patricide** (by our assumption in this case).
- **Conclusion:** In this world, the condition is **True** because of Oedipus.

This show that :

The OWA can lead to non-obvious conclusions through case-based reasoning. For instance, if we ask if Iokaste has a child who is a patricide and has a child who is not a patricide, DL can answer YES even if we don't know which child fits the description. This is because in all possible models, a specific individual (like Polyneikes) must either be a patricide or not, and in either scenario, the overall condition for Iokaste is satisfied

QUESTION 4 - LPAD

the candidate is invited to introduce the distribution semantics adopted in the LPAD. illustratin it with the use of a very simple exemple

Answare

In LPAD provides a framework for defining a probability distribution over a set of normal logic programs, referred to as "possible worlds".

Distribution Semantics in LPAD

The core idea of distribution semantics is that a probabilistic logic program (PLP) defines a probability distribution over several discrete "worlds". The probability of any given query is determined by summing the probabilities of all the worlds in which that query is true.

The different world are created by exactly one atom(one of the scenarios) from the head of each rule.

Illustrative Example: The Sneezing Bob

Consider a simple scenario where we want to determine the probability that a person, Bob, will sneeze based on two conditions: having the flu or having hay fever.

1. The LPAD Program: The knowledge is represented as follows, where null represents the case where the condition does not lead to sneezing:

- sneezing(X):0.7 ; null:0.3 :- flu(X). (A person with flu sneezes in 70% of cases).
- sneezing(X):0.8 ; null:0.2 :- hay_fever(X). (A person with hay fever sneezes in 80% of cases).
- flu(bob). (Bob has the flu).

- hay_fever(bob). (Bob has hay fever).

2. Generating Possible Worlds: By grounding the program for Bob, we get two probabilistic clauses. Selecting one atom from each head creates 2 * 2 = 4 possible worlds (w_1, w_2, w_3, w_4):

- w_1: sneezing(bob) from both rules. P(w_1) = 0.7 * 0.8 = 0.56.
- w_2: null from the first rule, sneezing(bob) from the second. P(w_2) = 0.3 * 0.8 = 0.24.
- w_3: sneezing(bob) from the first, null from the second. P(w_3) = 0.7 * 0.2 = 0.14.
- w_4: null from both rules. P(w_4) = 0.3 * 0.2 = 0.06.

3. Calculating the Query Probability: To find the probability of the query sneezing(bob), we identify all worlds where sneezing(bob) is true. In this case, it is true in w_1, w_2, and w_3.

P(sneezing(bob)) = P(w_1) + P(w_2) + P(w_3)

P(sneezing(bob)) = 0.56 + 0.24 + 0.14 = 0.94

Thus, the probability that Bob will sneeze is 0.94

QUESTION 5 – EC exercise

The candidate is invited to model the following situation using the Event Calculus approach.“A company produces a headlight for alpinists. There is only one button. Initially the headlight is off; when the button is pressed a first time, the light turns steady on; a further pressure of the button with the light steady on makes the light blinking; a further pression of the button switches the light off.

Answare

1. Ontology

First, we define the symbols that represent the elements of our system[22].

- Fluents (F):
    - off: The headlight is currently off.
    - steady_on: The headlight is emitting a steady light.
    - blinking: The headlight is in blinking mode.
- Events (E):
    - press: The action of pressing the button.

2. Initial State

The problem states that "Initially the headlight is off".

- Initially(off).

3. Domain-Dependent Axioms

We use the Initiates and Terminates predicates to describe how the press event changes the fluents based on the current state of the system.

Transition 1: From Off to Steady On

"When the button is pressed a first time, the light turns steady on."

- Initiates(press, steady_on, T) ← HoldsAt(off, T).
- Terminates(press, off, T) ← HoldsAt(off, T).

Transition 2: From Steady On to Blinking

"A further pressure of the button with the light steady on makes the light blinking."

- Initiates(press, blinking, T) ← HoldsAt(steady_on, T).
- Terminates(press, steady_on, T) ← HoldsAt(steady_on, T).

Transition 3: From Blinking to Off

"A further pression of the button switches the light off."

- Initiates(press, off, T) ← HoldsAt(blinking, T).
- Terminates(press, blinking, T) ← HoldsAt(blinking, T).

QUESTION 6 – Rete Algorithm

the candidate is invited to briefly introduce the RETE algorithm

Answare

The RETE algorithm, is a highly efficient pattern-matching algorithm designed for Production Rule Systems. It specifically optimizes the Match step of the reasoning process, which involves determining which rules' Left Hand Sides (LHS) are satisfied by the current facts in the working memory.

Core Concepts and Efficiency

The primary goal of RETE is to solve the "many patterns vs. many objects" matching problem. It achieves efficiency by avoiding repetitive iterations in two ways:

- Avoiding Iteration over Facts: Instead of re-evaluating rules from scratch whenever a fact changes, RETE stores the facts that match each pattern. When a fact is added, deleted, or modified, only the relevant matches are updated.
- Avoiding Iteration over Rules: The algorithm "compiles" the LHS of all rules into a specialized network of nodes, allowing the system to process changes across all rules simultaneously rather than checking them one by one.

The RETE Network Structure

The algorithm organizes patterns into a network consisting of two main types of memories:

1. Alpha Networks and Memories: These handle intra-element patterns (testing features within a single fact, like Age > 30). The results are stored in alpha memories.
2. Beta Networks and Memories: These handle inter-element patterns (testing relationships between different facts, such as matching a Professor's name to a Course's teacher). The results are stored in beta memories.

At the end of the network, the beta store contains the conflict set—a list of fully matched rules that are ready to be executed.

QUESTION 7 – LPAD exercise

The candidate is invited to represent, using LPADs, the following situation: A student X will pass FAIKR exam with probability 0.85 if she physically attends the lessons. A student X will pass the FAIKR exam with probability 0.70 if she watches the recordings. A student that is abroad will watch the lessons with probability 0.9. Chiara is abroad. The candidate must choose proper predicates and define the probabilistic rules. According to the Distribution Semantics, which is the probability that Chiara will pass the exam?

Answare

1. LPAD Representation

In an LPAD, the head of a clause is a disjunction where each disjunct is annotated with a probability. Based on the problem description, we can define the following rules:

- **Student X passes if they attend lessons (85% probability):**

pass(X) : 0.85 ; fail(X) : 0.15 :- attend_physically(X).

- **Student X passes if they watch recordings (70% probability):**

pass(X) : 0.70 ; fail(X) : 0.30 :- watch_recordings(X).

- **A student abroad watches recordings (90% probability):**

watch_recordings(X) : 0.9 ; null : 0.1 :- abroad(X).

- **Chiara is abroad:**

abroad(chiara).

2. Probability Calculation (Distribution Semantics)

According to the **Distribution Semantics**, the probability of a query is the sum of the probabilities of the "worlds" (or instances of the program) where the query is true.

Step 1: Identify the relevant ground rules for Chiara

1. abroad(chiara). (Certainty = 1.0)
2. watch_recordings(chiara) : 0.9 ; null : 0.1 :- abroad(chiara).
3. pass(chiara) : 0.70 ; fail(chiara) : 0.30 :- watch_recordings(chiara).

Note: Since there is no information about Chiara attending physically, the rule for attend_physically does not contribute to her passing.

Step 2: Define the Possible Worlds and their Probabilities We look at the combinations of choices for the probabilistic rules:

- **World 1 (Chiara watches and passes):**
    - abroad(chiara) is true.
    - Choice from Rule 2: watch_recordings(chiara) (Prob = 0.9).
    - Choice from Rule 3: pass(chiara) (Prob = 0.7).
    - **Probability of World 1:** 0.9 * 0.7 = 0.63.
    - **Result:** Chiara passes.
- **World 2 (Chiara watches and fails):**
    - Choice from Rule 2: watch_recordings(chiara) (Prob = 0.9).
    - Choice from Rule 3: fail(chiara) (Prob = 0.3).
    - **Probability of World 2:** 0.9 * 0.3 = 0.27.

- **Result:** Chiara does not pass.

- **World 3 (Chiara does not watch):**
  - Choice from Rule 2: null (Prob = 0.1).
  - In this world, watch_recordings(chiara) is false, so Rule 3 cannot be triggered. We have no information about success rate of students that doesn't watch the recordings
  - **Probability of World 3:** 0.1.
  - **Result:** Chiara does not pass.

Step 3: Total Probability The query pass(chiara) is true only in **World 1**.

P(pass(chiara)) = P(World_ 1) = 0.63

Final Answer:

The probability that Chiara will pass the exam is **0.63** (or 63%).

QUESTION 8 – Ontology Operation

Within the terminological approach towards the representation of concepts/categories and individuals/instances, the candidate is invited to illustrate the notions of

• *Disjointness over a set S of categories (S = {c1, c2, …, cn}, where c1…cn are*

categories)

• *Exhaustive Decomposition of a category c into a set S of categories*

• *Partition of a category c into a set of categories S*

The candidate is invited to illustrate these notions through a simple example

Answare

In the terminological approach to representing knowledge, categories and their relationships are defined using specific logical notions to ensure clarity and consistency within an ontology.

Below are the illustrations of the requested notions based on the provided material:

1. Disjointness

Definition: A set of categories S = {c_1, c_2, ..., c_n} is considered disjoint if they share no common members. Formally, an individual that belongs to one category in a disjoint set cannot belong to any other category in that same set.

- **Example**: The categories **Animals** and **Vegetables** are disjoint. While both are subcategories of "Living Things," an entity cannot be classified as both an animal and a vegetable simultaneously.
- **Formal notation**: Disjoint({Animals, Vegetables}).

2. Exhaustive Decomposition

Definition: A set of categories S provides an exhaustive decomposition of a parent category c if every instance of c must belong to at least one of the categories in S. Essentially, the subcategories "cover" the entire extent of the parent category.

- **Example**: If every person currently in a specific classroom is either a student or a professor, then the categories **Student** and **Professor** form an exhaustive decomposition of the category **PeopleInThisRoom**. Note that in an exhaustive decomposition alone, an individual could potentially belong to both (e.g., a student who is also a teaching assistant/professor).
- **Formal notation**: ExhaustiveDecomposition({Student, Professor}, PeopleInThisRoom).

3. Partition

Definition: A partition occurs when a set of categories S is both **disjoint** and forms an **exhaustive decomposition** of a parent category c. This means every instance of the parent category belongs to exactly one subcategory in the set—no more and no less.

- **Example**: Consider the category **Living Things**. If we follow a classification where every living thing must be either an **Eukaryote** or *Prokaryote*, and no living thing can be more than one of these, then { **Eukaryote**, *Prokaryote*}, forms a partition of **LivingThings**.
- **Formal notation**: Partition({ **Eukaryote**, *Prokaryote* }, LivingThings).

QUESTION 9 – Semantic web

The candidate is invited to briefly introduce the notion of Semantic Networks, and to highlight some of the limits that were present in their original formulation.

Answare

Semantic Networks (or semantic nets) are a form of knowledge representation where information is stored as a graph consisting of nodes, representing concepts or entities, and labeled arcs, which represent the relationships between those nodes. In modern contexts, such as Knowledge Graphs, these relationships are often expressed as "triplets" in the form of (subject, predicate, object) or (head, relation, tail).

While Semantic Networks provided a foundation for structured knowledge, their original formulations and early implementations (often associated with the early Web or "Web 1.0") faced several critical limitations:

- **Lack of Formal Semantics:** Original formulations often lacked a rigorous logical foundation, making it difficult for machines to automatically reason about the data or perform complex inferences.
- **Ambiguity in Relationships:** In early web structures, links existed between pages, but there was no explicit information regarding what the link itself represented (e.g., whether a link pointed to a friend, a colleague, or a related topic).
- **Non-Standardized Structure:** Early tags (like <title> in HTML) provided only implicit, non-structured semantics, and their use was not standardized across different systems.
- **Information Distribution and Inconsistency:** Because anyone could publish anything, information was often distributed, incomplete, or inconsistent across different sources, making it hard to integrate data into a single, reliable network.
- **Computational Complexity:** As networks grew to an industrial scale (like the "T-Box" in Description Logics), reasoning became computationally expensive and, in some cases, might not even terminate.

To overcome these limits, the **Semantic Web** introduced standards like **URIs** for unique identification , **RDF** for structured data exchange , and **OWL** to provide the formal logic necessary for automated reasoning

QUESTION 10 – Description Logic

The candidate is invited to briefly introduce the ALC Description Logics, mentioning the operators that are supported (negation, AND, ALL, EXISTS), and their meaning (possibly with a short example for each operator).

The candidate is invited to briefly introduce the Description Logics, and in particular the principal operators of the ALC fragment (AND operator; ALL operator; [EXISTS 1 r] operator; concept complement (negation)).

Answare

Description Logics used to represent knowledge through concepts, which represent categories of individuals, and roles, which represent binary relations between individuals.

The ALC fragment supports the following concept-forming operators:

1. Negation (NOT): Denoted by the symbol ¬, it represents individuals that do not belong to a specific concept. For example, ¬Patricide refers to individuals who are not patricides.
2. Intersection [AND d1…dn]: it describes individuals that belong to multiple categories simultaneously. For example, [AND Student Graduated] describes an individual who is both a student and has graduated.
3. Universal Quantification [ALL r d]: it refers to individuals who are related via a specific role (r) only to individuals of a certain class (d). For example, [ALL :HasChild Male] describes individuals who either have no children or have children who are all male.
4. Existential Quantification [EXISTS n r]: it represents the class of individuals related by a specific role (r) to at least n other individual. For example, [EXISTS 1 :HasChild] describes individuals who have at least one child.
5. [FILLS r c]: it refers to individuals who are related via a specific role (r) only to a specific individual (a constant). For example, [FILLS :Child francescoChesani] describes the set of individuals who have the specific person "francescoChesani" as their child.

(optional)

In ALC, a Knowledge Base consists of a TBox (terminological axioms defining the relationships between concepts) and an ABox (assertions about specific individuals). Reasoning in ALC typically focuses on subsumption, which determines if one concept is a subset of another, and satisfiability, which checks if a concept can actually have any individuals. Under the Open World Assumption, if a fact cannot be proven true, it is considered unknown rather than false.

QUESTION ? – Modal Logic

I don't found any questions about modal logic but there is the possibility that it can be part of the exam

Answare

Modal Logic is an extension of classical logic (such as propositional or first-order logic) designed to reason about concepts like necessity, possibility, knowledge, or time. While classical logic focuses on whether a statement is simply true or false, modal logic expresses propositional attitudes, or how an agent relates to a mental object (e.g., "Federico *knows* that there is a pizza in the fridge").

1. Fundamental Elements

Modal logic is built upon several technical pillars:

- Modal Operators: These are special operators that take entire sentences as input rather than regular terms. For example, the operator K_a P indicates that agent a knows P.
- Possible Worlds Semantics: This starts from the premise that an agent is not omniscient and considers multiple scenarios as "possible worlds" based on their current state.
- Kripke Structure (M): A formal model defined by:
    - S: A set of states or worlds.
    - pi: A function specifying which primitive propositions hold in each state.
    - K (Accessibility Relation): A binary relation where (s, t) in K means agent i considers world t possible from world s.
- Definition of Truth: An agent knows p if and only if p is true in every world accessible from the current one.

2. Axioms and Inference Rules

The specific behavior of a modal logic depends on the axioms and rules it adopts:

- Axiom 0 (A0): All instances of propositional tautologies are valid.
- Modus Ponens (MP): An inference rule stating that if phi is valid and phi -> psi is valid, then psi is also valid.
- Knowledge Axiom (A2): K_i phi -> phi. This states that if an agent knows something, it must be true in the real world.
- Positive Introspection (A3): K_i phi -> K_i K_i phi. The agent is aware of what they know.
- Negative Introspection (A4): neg(K_i) phi -> K_i neg(K_i) phi. The agent is aware of what they do not know.

- By combining these, different logics are formed, such as Logic K (using A0, A1, MP, and G) or Logic S5 (using all axioms A0–A4, MP, and G).