

# Kesani Theory Questions

## Event Calculus Framework. Define its predicates:

Event Calculus framework is composed of an ontology and two different sets of axioms.

- EC ontology (fixed)
- Domain independent axioms (fixed)
- Domain dependent axioms

Ontology predicates:

- **HoldsAt**(F, T): the fluent F holds at the time T.
- **Happens**(E, T): the event E (the fact that an action has occurred) happens at the time T.
- **Initiates**(E, F, T): the event E causes the fluent F to hold at time T.
- **Initially**(F): the fluent F holds at time 0.
- **Terminates**(E, F, T): the event E causes the fluent F to terminate (cease to hold) at time T.
- **Clipped**(T1, F, T): the fluent F has been made false between T1 and T, s.t.  $T1 < T$ .

## (Upper ontologies) Define the following notions, providing an example for each of them:

- Disjointness over a set S of categories ( $S = \{c_1, c_2, \dots, c_n\}$ , where  $c_1 \dots c_n$  are categories)

$$Disjoint(s) \leftrightarrow \{\forall c_1, c_2 \in s \wedge c_1 \neq c_2 \rightarrow c_1 \cap c_2 = \emptyset\}$$

*Example:* Disjoint ({Smartphones, Fruits}).

- Exhaustive Decomposition of a category c into a set S of categories

$$ExhaustiveDecomposition(s, c) \leftrightarrow \{\forall i \in c \exists c_1 \in s \wedge i \in c_1\}$$

*Example:* ExhaustiveDecomposition ({Oranges, Lemons}, Citrus).

- Partition of a category c into a set of categories S

$$Partition(s, c) \leftrightarrow ExhaustiveDecomposition(s, c) \wedge Disjoint(s)$$

*Example:* Partition ({Plants, Animals}, LivingThings).

## 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.

Semantic Networks is a **graphical** language that describes the relationship between categories and objects (and their properties), and the reasoning among them. Categories and objects are represented by oval or boxes, and they are connected by links. These links can be also labelled: this label defines the nature of the link itself.

In semantic networks we can represent two types of properties:

- Properties about the categories (as a whole).
- Properties about the members of the categories.

#### Limitations:

- **Multiple inheritance** allowed: this can cause some problems and incoherencies, like the Diamond problem (an object belongs to two semantically different categories).
- **Less powerful** than FOL

#### Advantages:

- **Simplicity** (in representation).
- **Defaults** properties: they can be directly assigned to the categories.
- **Procedural attachment**: Semantic Networks can attach a call to a special procedure when some special cases, that have to be treated differently, are encountered.

The candidate is invited to introduce the notions of Close World Assumption and Open World Assumption, and to briefly discuss how Prolog and Description Logics deal with these aspects.

Because **Prolog** allows only definite clauses (then no negative literals), and since SLD does not allow to derive negative information, negation represents a problem.

With negation, we mean that we want to derive if something is **not** a logical consequence of something else. *E.g.* if A is not in the KB, or A then we know that A is not true in the KB. But we are unsure about its truth value. With CWA, everything we don't know is False. In other words, everything that isn't explicitly stated in the KB is assumed to be False.

This can be resumed with *"If a ground atom A is not logical consequence of a program P, then you can infer  $\sim A$ "*.

CWA is too powerful: since FOL is semi-decidable, there is no algorithm that establishes (so terminates with a response) in a finite time if A is **not** a logical consequence of P. This means that SLD resolution is not guaranteed to terminate.

In **Description Logic**, unlike other logics, there is an Open World Assumption. *All of which that cannot be inferred, is unknown.*

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).

ALC Description Logics is an extended version of AL (Attributive Language) logics, which includes the negation of concepts.

The concept-forming operators are logical symbols (with a fixed meaning), and they are: AND, ALL, EXISTS, and the negation. They allow to express **complex concepts** combining atomic concepts together.

- **[ALL  $r\ d$ ]**: those individuals in the domain that are  $r$ -related *only* to the individuals of class  $d$ .
  - Example: [ALL :HasBikes Red] all the individuals that have 0 or more bikes, but they are all red.
- **[EXISTS  $n\ r$ ]**: those individuals in the domain that are  $r$ -related to *at least*  $n$  other individuals.
  - Example: [EXISTS 2 :HasBikes] all the individuals that have at least 2 bikes.
- **[AND  $d_1\dots d_n$ ]**: anything described by  $d_1\dots d_n$ . Every individual is a member of all the categories  $d_1\dots d_n$ . This is the idea of **intersection**.
  - Example: [AND Child Male] an individual that is both a child and a male.

[Optional] Additional DL operators:

- **[FILLS  $r\ c$ ]**: those individuals that are  $r$ -related to individuals identified by  $c$ .
  - Example: [FILLS :HasChild gino] individuals that have gino as child.
- **[AT-MOST  $n\ r$ ]** describes individuals related by role  $r$  to at most  $n$  individuals.
  - Example: [AT-MOST 3: Child] individuals that have at most 3 children.
- **[ONE-OF  $c_1c_2 \dots c_n$ ]** is a concept satisfied only by  $c_i$ , used in conjunction with ALL would lead to a restriction on the individuals that could fill a certain role.
  - Example: (Beatles  $\triangleq$  [ALL :BandMember [ONE-OF john paul george ringo]])

The candidate is invited to briefly introduce the three different approaches (presented in the course), to deal with the reasoning with temporal information.

**Situation Calculus** represents the dynamicity of the world considering the propositions that describe the world, called *fluents*. In other words, a fluent indicates if a property holds in a given situation. A situation is a “snapshot” of the world describing properties (the fluents) that holds in that given situation (state).

**Event Calculus** focuses on points in time. In contrast with Situation Calculus, it reifies situations and fluents into terms. Fluents are now properties whose truthness value changes dynamically, over time. The frame problem is resolved explicating some predicates in its ontology.

**Allen’s logic** focuses on intervals of time, which begin at a certain point in time and terminates in another one. Allen assumes that events aren’t instantaneous, but he represents them with a **duration**. This is a much more expressive approach (rather than Event Calculus), but it’s very difficult to use this in a reasoning system.

The candidate is invited to shortly introduce rule-based forward reasoning, and to briefly highlight the main difference w.r.t. backward reasoning.


**Backward reasoning** is a goal-oriented reasoning process. It starts from the goal and iteratively works backwards to find a sequence of steps that lead to the goal. “Logically” speaking, this results in searching a proof that finds the facts that make the given conclusion true.

The main **difference** between backward reasoning and forward reasoning regards the situation **new facts** are added to the Knowledge Base. Normally, in Backward Reasoning, when this situation occurs, the reasoning mechanism must be restarted.

**Forward Reasoning**, instead, adopt the Production Rule approach. When new facts are available, they can be dynamically added to the KB. When new facts match the LHS of any rule, the reasoning mechanism is **triggered**, until it reaches **quiescence** again. This can be viewed as a more *procedural* framework, rather than a logical one.

Therefore, side-effects in the RHS are allowed: the **working memory** can be modified inserting or deleting facts.

The main **steps** of a forward reasoning mechanism when a new fact is added to the KB are the following:

- 
1. **Match**: search for the rules whose LHS matches the fact and decide which ones will trigger.
  2. **Conflict resolution / activation**: triggered rules are put into the *Agenda*, and possible conflicts are solved.
  3. **Execution**: RHS are executed, effects are performed, the KB is modified, etc.
  4. **Go back to step 1**.

The candidate is invited to shortly introduce rule-based forward reasoning and the RETE algorithm.

[see previous question for Forward reasoning]

RETE algorithm focuses on the **matching step** of a forward reasoning mechanism. The matching step computes the rules whose **LHS** is matched with the new fact. We can say that it is a "*many patterns*" vs "*many objects*" pattern match problem.

**LHS matching / satisfaction**: since LHS is usually expressed as a conjunction of **patterns**, this step consists of verifying if all patterns are matched, then if they have a corresponding element (a fact) in the working memory.

The problem is how to determine in an efficient way which are the rules that should be **triggered**. Doing this iteratively results in a cost that is linear with the dimension of facts.

RETE algorithm **avoids** the **iteration over facts** determining the so called "**conflict set**": for each pattern, it stores the facts that match it.

- When a new fact is added / deleted / modified, all the patterns are compared and the list of matches, the conflict set, is updated.

RETE algorithm also **avoids** the **iteration over rules**. It is accomplished compiling a LHS into a network of nodes. For a LHS, we define **two types of patterns**:

- Patterns testing **intra-elements** features: they are computed into **alpha-networks**, and their outcomes are stored into **alpha-memories**.
- Patterns testing **inter-elements** features: computed into **beta-networks**, while their outcomes are stored into **beta-memories**. At the end, **beta-memory** contains the conflict set.

The candidate is invited to briefly introduce the Knowledge Graph paradigms, and which are the main differences w.r.t. the Semantic Web proposal.

Knowledge Graph proposal is basically different from the Semantic Web one.

**Semantic Web's** proposal exploits Description Logic (either RDF or OWL) to reason upon the available data, extending the web with semantic knowledge, a type of knowledge that concerns the content. In order to achieve this, Semantic Web reasons on the **T-Box**, that contains axioms and sentences which describe the concepts. But doing this is too **complex** and **expensive**: this process is **slow**, and its termination is **uncertain**.

On the other hand, **Knowledge Graph** proposal is statistical proposal. With KG, A-box (lists of facts) and the T-Box are considered at the same level, so **there is no reasoning at all!** Axioms are ignored and billions of facts are considered instead. Therefore, there is no conceptual schema: data with different semantics can be mixed, avoiding inconsistencies. Since Knowledge Graphs are not based on Description Logic, they are less formal and less descriptive, but they gain enormous advantages regarding **computational speed**.

The **idea** behind KG is simple: the information is stored inside nodes and there are arcs connecting them. This means that **knowledge** is represented through a **graph** where the nodes are terms, and the edges are the relations between them. Given that, **queries** can be solved by exploring the graph using traditional graph algorithms.

The candidate is invited to describe how negation is tackled in Prolog, what is NAF, what is SLDNF, and the issues related with NAF over terms containing unbounded variables.

To accomplish negation, Prolog restricts **CWA** to those atoms whose proof terminates in finite time using the Negation as Failure. **NAF** derives only the negations of atoms whose proof does terminate in a finite time. We'll call this special set of atoms  $FF(P)$ . Therefore, the NAF rule says:

$$NAF(P) = \{\sim A \mid A \in FF(P)\}$$

This rule says that **if A belongs to FF(P), then A is not a logical consequence of P**. But this is not always true in the other direction, because not all the atoms that are not logical consequence of P belong to FF(P).

**SLDNF** is used in Prolog to implement NAF. It extends SLD resolution with the NAF rule and it behaves in the following way:

- We want to prove  $\sim A$  (A is an atom) and the prolog interpreter tries to prove A:
  - o If the proof for A succeeds, then the proof for  $\sim A$  fails.
  - o Otherwise, if the proof for A fails in a finite time, then the proof of  $\sim A$  succeeds.

Prolog does not check if the selected literal is ground or not. If a non-ground (unbounded) literal is selected, SLDNF provides an incorrect derivation. This is due to the quantification of the variables appearing in the negative literal. Our goal is to check if there exist an X such that  $p(X)$  is False. But in SLDNF this is translated with all the X s.t.  $p(X)$  is false.

The candidate is invited to describe the distribution semantics adopted in LPAD, also using a short program to illustrate such semantics.

**LPAD** (Logic Programs with Annotated Disjunctions) is one of the several **PLP** languages.

A PLP (Probabilistic Logic Programming) language defines a probability distribution over normal logic programs, called **worlds**.

In LPAD the head of a clause is extended with disjunctions, and each disjunct is annotated with a probability.

*Example:*

- $toothache(X):0.8 ; null(X):0.2 : - cavity(X)$
- $toothache(X):0.6 ; null(X):0.4 : - wisdom\_tooth(X)$
- $cavity(armando).$
- $wisdom\_tooth(armando).$

We notice that each rule has a probability distribution over its head.

We can obtain worlds grounding the program and then decide if, for each atom in each head, to include it or not (then selecting the null).

The **probability** of a **world w** is given by:

$$P(w_\sigma) = P(\sigma) = \prod_{(C,\theta,i) \in \sigma} P(C,i)$$

Where:

- $\sigma$  is a selection of one atomic choice for every grounding of each clause.
- A selection  $\sigma$  identifies a logic program  $w_\sigma$  called world.
- $C$  is a clause,  $\theta$  is a substitution s.t.  $C\theta$  is **ground**.
- An **atomic choice** is a selection of the  $i$ -th atom of the head of  $C$  for grounding  $(C, \theta, i)$ .

For example, the probability of the world  $w_1$  where:

- $toothache(armando):0.8 : - cavity(armando)$
- $toothache(armando):0.6 : - wisdom\_tooth(armando)$
- $cavity(armando).$
- $wisdom\_tooth(armando).$

$$P(w_1) = 0.8 \times 0.6 = 0.48.$$

### [Extra] Modal Logics, resume:

Until the introduction of Modal Logics, the agents are not able to reason upon their own knowledge. This means that they do **not** perform any type of **meta-reasoning** and they are not capable of distinguishing between the different flavors of knowledge, **the propositional attitudes**: the knowledge itself, the belief, the perception, etc. Moreover, an agent with a KB also doesn't know anything about the knowledge of the other agents in the world.

These are all limitations which have been overcome by **Modal Logics**. Since FOL is powerful but still it's not the best choice to represent propositional attitudes, Modal Logics maintains FOL's syntax but augments its operators introducing special ones, the **modal operators**, useful to represent the **propositional attitudes**.

Concerning semantics, it includes the notion of possible worlds, worlds that are reachable from the current one through an accessibility relation. They represent the **unknown**: since an agent is not omniscient, it *knows*  $p$  if in any accessible world accessible from the current,  $p$  is true.

Formally, the semantics of Modal logics is given by a *Kripke* structure:  $M = \{S, \pi, K_1 \dots K_n\}$  where:

- **S** represents the set of states of the world.
- $\pi$  specifies in which state the given proposition holds.
- $K_1, \dots, K_n$  are binary relations with the meaning  $(s, t) \in K_i$  if the agent  $i$  considers the world  $t$  possible (reachable) from  $s$ .

The fundamental operator **K** indicates that an agent  $a$  **knows**  $P$ :

$$K_A P$$

It's important to notice that Modal Logic's framework has been exploited to support a **qualitative** representation and reasoning over the **evolution** of worlds along the **temporal** dimension.

### [Extra] Complex Event Processing, resume:

### [Extra] Drools, resume: