0) The candidate is invited to introcude the vanilla meta-interpreter, and to explain its clauses.

[Quesstion found in every exam]

The vanilla meta-interpreter for the Prolog language can be define as follow:

```
solve(true)   :- !.
solve((A, B)) :- !, solve(A), solve(B).
solve(A)      :- clause(A, B), solve(B).
```

The vanilla meta-interpreter explores the current Prolog program, searching for the clauses until it can prove the goal or it fails. It is defined as a predicate 'solve(goal)' that answers true if the goal can be proved using the clauses of the current program. In particular:

- the first clause tells us that -> a tautology is a success;

- the second clause tells us that -> to prove a conjuntion we have to prove both atoms (following left-most rule);

- the third clause tells us that -> to prove an atom A, we look for a clause A :- B that has A as conclusion and prove its premise B.

We can easly edit this meta-interpeter. For example, if we want to change the rule of how the meta-interpreter select the subgoal in the current resolvent from left-most to right-most... we can write:

```
solve(true)   :- !.
solve((A, B)) :- !, solve(B), solve(A).
solve(A)      :- clause(A, B), solve(B).
```

In addition, if we want to get the number of steps needed to prove the goal... we can write:

```
solve(true, 0)   :- !.
solve((A, B), S) :- !, solve(B, SB), solve(A, SA), S is SB + SA.
solve(A, S)      :- clause(A, B), solve(B, SB), S is 1 + SB.
```

Finally, if we wanto to print the subgoal before and after its execution... we can write:

```
solve(true, 0)   :- !.
solve((A, B), S) :- !, solve(B, SB), solve(A, SA), S is SB + SA.
solve(A, S)      :- write('Solving: '), write(A), nl,
                    clause(A, B),
                    write('Selected Rule: '), write(A), write(':-'), write(B), nl
                    solve(B, SB), S is 1 + SB,
                    write('Solved: '), write(B), nl.
```

---

1) 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 Description Logics.

[Question found in the exams: 15/01/2025 (tracks A and B), 12/09/2023, 09/02/2023]

The Closed-World Assumption (CWA) states that if a ground atom A is not logical consequence of a program P, the you can infer ~A:

CWA(P) = { ~A | it does not exists a refutation SLD for P ∪ {A} }

Thus, if something is not represented in the knowledge base, then you can infer that it is false. In addition, CWA is a non-monotonic inference rule, meaning that adding new axioms to the program might change the set of theorems that previously held.

Due to FOL (First-Order Logic) undecidability, there is no algorithm that establishes in a finite time if A is not logical consequence of P. For this reason, Prolog drops the CWA and it choses to implement the Negation as a Failure (NF). This rule derives only negation of atoms whose proof terminates with failure in a finite time. Given a program P and the set of atoms for which the proof fails in a finite time, FF(P), then this rule can be defined as:

NF(P) = { ~A | A ∈ FF(P) }

However, it is worthy to say that not all the atoms that are not logical consequence of P belong to FF(P).

Finally, Prolog uses NFSLD (SLD resolution + NF) to solve goals containing also negative atoms, and we can ensure the correctness and the completness of this rule by selecting only positive or ground-negative literals.

The Open-World Assumption (OPW) states that if a sentence cannot be inferred from the knowledge base, then its truth value is unknown and it is not correct to assume that it is false. Description Logics (DL) are based on the Open-World Assumption.

DL handle the OWA by treating missing information as unknown rather than false, hence it remains open to the possibility of that information of being true or false. Reasoning in DL under OWA involves considering all possible models that satisfy the known information in the KB. When encountering unknown facts, the reasoner splits the possible models into different scenarios depending on the various ways the fact could hold true or false. This approach allows DL to represent and reason about inpmplete knowledge.

---

2) The candidate is invited to introduce the Event Calculus framework, by breafly describing the terms (a.k.a. the EC ontology) and the axioms.

[Question found in the exams: 15/01/2025 (track A), 12/09/2023, 16/01/2023, 21/12/2022]

The Event Calculus framework (Kowalski's formulation) was proposed in 1986 to enable qualitative time reasoning about the execution of actions, events happening and how such events change the environment. It allows to represent the state of a system as (FOL) logical terms. Moreover, it allows to represent and reason on how a system evolves as a consequence of happening events.

Specifically, Event Calculus (EC) reifies both fluents (properties) and events (actions) into terms (instead of predicates as in Situation Calculus). EC is based on points of time (instead of intervals as in Allen's Logic), and the fluents are seen as properties whose truthness value changes over time.

Its formulation comprises an ontology and two distinct set of axioms.

- EC ontology it is a set of fixed reserved keywords:
    * initially (F   ): The fluent F holds at time 0.
    * holdsAt   (F, T   ): The fluent F holds at time T.
    * happens   (E, T   ): The event E (i.e. the execution of an action) happened at time T.
    * initiates (E, F, T): The event E causes the fluent F to start holding at time T.
    * terminates(E, F, T): The event E causes the fluent F to cease holding at time T.
    * clipped   (T', F, T): The fluent F has been made false between time T' and T, where T' < T.

- EC domain-independent axioms are a fixed set of axioms describing how fluents change:
    * Truthness of a fluent:
        1. $holdsAt(F, T) \Leftarrow happens(E, T') \wedge initiates(E, F, T') \wedge (T' < T) \wedge \neg clipped(T', F, T)$
        2. $holdsAt(F, T) \Leftarrow initially(F) \wedge \neg clipped(0, F, T)$
    * Clipping of a fluent:
        1. $clipped(T', F, T'') \Leftarrow happens(E, T) \wedge (T' < T < T'') \wedge terminates(E, F, T)$

- EC domain-dependent axioms are a collection of axioms specific for a certain domain, defined using the predicates: initially, initiates and terminates.

---

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

[Question found in the exams: 15/01/2025 (track B), 17/01/2024 (tracks A and B), 12/06/2023]

The Event Calculus framework (Kowalski's formulation) was proposed in 1986 to enable qualitative time reasoning about the execution of actions, events happening and how such events change the environment. It allows to represent the state of a system as (FOL) logical terms. Moreover, it allows to represent and reason on how a system evolves as a consequence of happening events.

Specifically, Event Calculus (EC) reifies both fluents (properties) and events (actions) into terms (instead of predicates as in Situation Calculus). EC is based on points of time (instead of intervals as in Allen's Logic), and the fluents are seen as properties whose truthness value changes over time.

Its formulation comprises an ontology and two distinct set of axioms.

- EC ontology it is a set of fixed reserved keywords:
    * initially (F   ): The fluent F holds at time 0.
    * holdsAt   (F, T   ): The fluent F holds at time T.
    * happens   (E, T   ): The event E (i.e. the execution of an action) happened at time T.
    * initiates (E, F, T): The event E causes the fluent F to start holding at time T.
    * terminates(E, F, T): The event E causes the fluent F to cease holding at time T.
    * clipped   (T', F, T): The fluent F has been made false between time T' and T, where T' < T.

- EC domain-independent axioms are a fixed set of axioms describing how fluents change:
    * Truthness of a fluent:
        1. holdsAt(F, T) ⇐ happens(E, T') ∧ initiates(E, F, T') ∧ (T' < T) ∧ ¬clipped(T', F, T)
        2. holdsAt(F, T) ⇐ initially(F) ∧ ¬clipped(0, F, T)
    * Clipping of a fluent:
        1. clipped(T', F, T'') ⇐ happens(E, T) ∧ (T' < T < T'') ∧ terminates(E, F, T)

- EC domain-dependent axioms are a collection of axioms specific for a certain domain, defined using the predicates: initially, initiates and terminates.

Event calculus only captures instantaneuous events that happen at a given points in time. In contrast, Allen's logic allows to reason about intervals of time. An interval, i, starts at a certain time point, indicated by the function begin(i), and it ends at a certain time point, indicated by the function end(i). To represent durative event in terms of start and end time points, an origin and a measurment unit must be defined in Allen's logic and points in time are measured with respect to the distance from the origing. Furthermore, instantaneous events are still representable and we say that they are durative events with duration zero.

In Allens'logic has been defined several temporal operators:

- after   (j, i) ⇔ before(i, j)

- meet   (i, j) ⇔ end(i)   = begin(j)

- finishes(i, j) ⇔ end(i)   = end(j)
- starts (i, j) ⇔ begin(i) = begin(j)

- before (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)

- equals (i, j) ⇔ starts(i, j) ∧ ends(i,j)

Modal Logics (ML) have the same syntax as FOL, but modal operators are introduced. ML are based on the idea of interacting agents, each with its own knowledge base. However, an agent is not able to distinct between knowledge, belief or perception there is possibility of any meta-reasoning on its own knowledge. In addition, FOL predicates are not suited to represent proposition attitudes such us: believes, knows, wants or informs. This is the reason why modal operators are needed.

The modal operators take two inputs: the name of the agent and the sentence (instead of term) it refers to. The semantics is extended with the notion of possible worlds related through an accessiblity relation. For example, ML define the knowledge operator (Ka(P)) which states that an agent 'a' knows P.

An agent in a given scenario is not omnisceient, i.e. it has a current perception of the current world and considers the unknown as other possible worlds. For an agent, if P is true in any accessible worlds (from the

current one), then the agent knows P. Fromally, the semantics is defined by means of a set of primitive propositions ϕ and a Kripke structure M = (S, π, K1...Kn) where:

- S is the set of states of the world,

- π specifies in which states each primitive proposition holds,

- Ki is a binary relations stating that if (s, t) ∈ Ki then agent 'i' considers the world 't' possible from state 's'.

Finally, Modal Logics define the following axioms:

- Tautology: all propositional tautologies are valid.

- Modus Ponens: if ϕ and $\phi \Rightarrow \psi$ are valid, then ψ is valid.

- Distribution axiom: an agent can compute all possible consequences of its knowledge base (i.e. knowledge is closed under implication).

- Knowledge Generalization rule: an agent knows all the tautologies, i.e. for every M, if M |= ϕ then M |= Ki(ϕ).

- Knowledge axiom: if an agent knows ϕ, then ϕ is true.

- Introspection axioms: each agent has perfect knowledge about its own knowledge since (positive axiom) if the agent knows ϕ then it knows to know ϕ and (negative axiom) if the agent does not know ϕ then it knows to not know ϕ.

The framwork of modal logic has been exploited to support a qualitative representation and reasoning over the evolution of worlds alogn the temporal dimension. Time is discrete and each world is labeled with an integer. The accessbility relation now is mapped inot the temporal dimesion with two possible ways to evolve:

- Linear-time: from each world, there is only one other accessible world.

- Branching-time: from each world, there are many accessible worlds.

Linear-time Temporal Logic (LTL) consider the first way to evolve from a world to another and it defines the following operators:

- next $\bigcirc\phi$ -> ϕ is true in the next time step.
- globally $\Box\phi$ -> ϕ is always true from now on.
- future $\Diamond\phi$ -> ϕ is sometimes true in the future. This is equivalent to $\neg\Box(\neg\phi)$.
- until $\phi U\psi$ -> there exists a moment (now or in the future) when ψ holds. ϕ is guaranteed to hold forom now until ψ starts to hold.
- weak until $\phi W\psi$ -> there might be a moment when ψ holds. ϕ is guaranteed to hold forom now until ψ possibly starts to hold.

The sematic of the LTL operators is defined from a Kripke structure M.

LTL is often used for the modelling of system specifications that can be considered as finite state machines. LTL is an effective logic in modelling distributed systems as well, where different agents can exchange messages and information.

---

4) 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 (i.e. the negation)) possibly with a short example of each operator.

[Question found in the exams: 12/06/2023, 09/02/2023]

Description Logics (DL) is a family of logics that fosu on the description of the terms. The main objectives of DL is to provide a method to represent individuals, categories and roles, to infer hierarchies beetween concepts, to determine if an individual belongs to some category or not, and to represent complex concepts as the result of some composition of simpler concepts.

Description Logics define two different sets of symbols:

- logical symbols (with a fixed meaning):
    * punctuation ((,), [,])
    * positive integers
    * concept-forming operators (ALL, EXISTS, FILLS, AND)
    * connectives ($\sqsubseteq$, =, $\longrightarrow$)
- non-logical symmbols (domain-dependent):
    * atomic concepts
    * roles
    * constants

Complex concepts can be created by combining atomic concepts by using the concept-forming operators. A well-formed concept follows the conditions:

- every atomic concepts is a concepts

- if 'r' is a role and 'd' is a concept, then [ALL r d] is a concept

- if 'r' is a role and 'n' is a positive integer, then [EXISTS n r] is a concept

- if 'r' is a role and 'c' is a constant, then [FILLS r c] is a concept

- if d1...dn are concepts, then [AND d1...dn] is a concept

These concept-forming operators have the following meanings:

- [ALL   r d]: all the individuals that are r-related only to individuals of class d.

    * Example: [ALL :HasChild Male] is the class of individuals with zero children or only male children.

- [EXISTS n r]: exists a class of individuals in the domain that are r-related to at least n other individuals.

    * Example: [EXISTS 1 :Child] class of individuals with at least one child.

- [FILLS r c]: all the individuals r-related to the individual c.

    * Example: [FILLS :Child jhon] class of individuals with child 'jhon'.

- [AND d1..dn]: all the individuals that belogns to all the categories d1..dn.

    * Example: [AND Veichle [EXISTS 4 :Wheel]] the class of all the veichles with at least four wheels.

In Description Logics the knowledge base in any collection of sentences, i.e. expressions that are intended to be true or false in the domain. In particular:

- constant are considered to be individuals (i.e. the entities of our domain)

- concepts are classes or categories of individuals

- roles are binary relations over individuals

and each sentence follows these forms:

- if d1 and d2 are concepts, then d1 ⊑ d2 is a sentence

- if d1 and d2 are concepts, then d1 = d2 is a sentence

- if c is a constant and d is a concept, then c ⟶ d is a sentence

These connectives have the following meanings:

- d1 ⊑ d2: every individual that satisfies d1 satisfies also d2.

    * Example: PhDStudent ⊑ Student means that every PhD student is a student (not vice-versa)

- d1 = d2: individuals satisfying d1 are precisely those that satisfy d2.

    * Example: PhDStudent = [AND Student Graduated HasFunding] means every PhD student is a student, graduated and has funding (and vice-versa)

- c ⟶ d : the individual c satisfies the description expressed by concept d.

    * Example: federico ⟶ Professor means the individual 'federico' is an instance of category Professor

DL is a family of logics and different logics can be defined depending on which operators are admitted or not. Of course, more operators means:

- higher exressivity;

- higher computational costs;

- logic might be undecidable.

A minimal logic is named Attributive Language (AL) and includes:

- atomic concepts

- universal concept (Thing, T)

- bottom concept (Nothing, ⊥)

- atomic negation (¬A)

- AND, ALL, [EXISTS 1 r] operators

The Attrivutive Language Complement (ALC) is a logic that extends AL by adding the negation for concepts (i.e. the complement of a concept). For example, the complement of the class Dog is the class of every other individual that is not a Dog.

---

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

[Question found in the exams: 16/01/2023]

A Semantic Network is a graphical language used for representing obects and categories by their name (a lable) into ovals connected by links (also labelled). The link labels are used to characterise the nature of a link. There are four different types of links available:

1. To represent relations between objects
    * Example (SisterOf): <Mary> ⟶ SysterOf ⟶ <Jhon>
2. To represent a property of a category
    * Example (Legs): <Persons> ⟶ Legs ⟶ 2
3. To represent a "is-a" relation
    * Example (SubsetOf): <FemalePersons> ⟶ SubsetOf ⟶ <Persons>
4. To represent a propery of the members of a category
    * Example (HasMother): ∀ x, x ∈ Persons ⇒ [∀y, HasMother(x,y) ⇒ y ∈ FemalePersons]

Semantic Networks offer two type of reasoning:

1. Single inheritance reasoning: starting from an object, check if it has the queried property. If not, iteratively move up to the category it belogns to and check for the property.

2. Multiple inheritance reasoning: reasoning is not possible as it is not clear which parent to choose ("The diamond problem").

The advantage of using semantic networks is that it is easy to attach default properties to categories and override them on the objects. However, Semantic Networks suffer from some limitations. Compared to FOL (First-Order Logic), Semantic Networks do not have:

- negation

- universal and existential quantified properties

- disjunction

- nested function symbols

Many Semantic Network systems allow to attach sepcial procedures to handle special cases that the standard inference algorithm cannot treat. This approach is powerful and easay to be used but we losse declarative logical meaning.

(extend answer with notion of Frames)

Frames are pieces of knowledge used to describe objects in terms of its properties. In particular, each Frame has:

- a unique name

- properties represented as paris <slot-filler>

For example:

```
(
     toronto
          <:Instance-Of City>
          <:Province ontario>
```

```
          <:Population 4.5M>
)
```

An object represented using Frames belongs to a category if it is similar enough to some typical members of that category. Such members are called prototypes. Prototypes are useful to reason about general properties but defeasable ovalues are allowed. Moreover, slots can contain additional information about the fillers, named facets (e.g. default value), and fillers can be procedural attachments:

- if-needed: looks for the value of the slot
- if-added: adds a value
- if-removed: removes a value.

---

6) The candidate is invited to briefly introduce the RETE algorithm.

[Question found in the exams: 17/01/2024 (tracks A and B)]

The simplest rule has the form p1...pn ⇒ q1...qn where the Left Hand Side (LHS) contains the body of the rule (p1...pn) and the Right Hand Side (RHS) contains the head of the rule (q1...qn). The default reasoning mechanism is Modus Ponens: if the premises, A, are true and A ⇒ B is true, then the consequences, B, are true. This approach is effective to reason upon static knowledge and it is called backward chaining (e.g. Prolog).

With forward chaining we want to reason upon a dynamic knowledge base. Using the "production rule" approach new facts can be dynamically added to the knowledge base. When a fact is added, the reasoning mechanism is triggered:

a. Match step: search for the rules whose LHS match the fact and decide which to trigger.

b. Conflict Resolution/Activation step: triggered rules are put into the Agenda where possible conflicts are solved.

c. Execution step: the RHS of the triggered rules are executed and the effects are performed; the knowledge base is updated with the copies of the new facts.

These steps are executed until quiescence as the execution step may add new facts.

At the base of any Production Rule system there is the working memory, a data structure containing the current set of facts and the set of rules. The efficiency of the working memory determines the performance of a production rule system.

The RETE algorithm is an efficient algorithm for implementing rule-based systems, and it focuses on the "Match step". The LHS of a rule is usually expressed as a conjunction of patterns. Deciding if the LHS of a rule is satisfied consists on verifying if all the patterns do have a corresponding fact in the working memory.

Since matching has a linear complexity, RETE define a "conflict set", i.e. the set of all possible instantiations of the production rules. Each rule is described as: <Its RHS, list of facts matched by its LHS>. This method avoids iterations over facts by storing for each pattern which are the facts that match its LHS. In addition, we can compile the LHS into a network of nodes and considering two types of patterns:

1. testing intra-elements features: compiled into alpha networks; outcome is stored into alpha-memories and used by beta-networks.

2. testing inter-elements features: compiled into beta-networks; outcome is stored into beta-memories and corresponds to the conflict set.

For the "Conflict Resolution step", the RETE algorithm allows different strategies to handle conflicts:

- rule priority

- rule ordering

- temporal attributes

- rule complexity

and the best approach depends on the use case.

For the "Execution step", RETE exectutes all the rules in the Agenda and then checks for possible side effects that modify the working memeory in a second moment. However, it is possible to modify this behaviour.

---

7) The candidate is invited to briefly introduce the concept of Business Process Management.

[Possible question]

Business Process Management (BPM) is includes concepts, methods, and tecniques to support the design, administration, configuration, enactment, and analysis of Business Processes (BPs) by mining data contained in information systems. BPM became crucial since BPS became als a technical issue. In fact, whatever can be done automatically must be done automatically and then decisions are automatically taken.

The following business process lifecycle can be define:

- Design&Analysis: definition of conceptual schemas, process models, and then validation, simulation and properties verification.
- Configuration: implementation of the business process.
- Enactment: the process runs and event logs are collected for monitoring and future predicitons.
- Evaluation: process mining and assess the quality of the process.

Two types of buiness process can be defined:

- Organizational vs Operational
    * Organizational: process described by means of inputs, outputs, expected outcomes, and dependencies using (semi-)formal languages.
    * Operational: process described by means of activities (and relationships between them) while implementation is disregarded.
- Intra-organizational vs Inter-organizational

* Intra-organizational: all activities are exectued within the business boundaries

* Inter-organizational: some activities are exectued outside the business boundaries with no control over them.

A business process can also vary because of its degree of automation, degree of repetition or degree of structuring.

---

8) The candidate is invited to briefly introduce the concept of Petri Nets.

[Possible question]

Petri Nets are an abstract formalism for representing processes in a graphical way. Petri Nets are both formal (semantics well-defined and not ambiguous) and abstract (no info about the execution environment) and are able to model dynamic sustems with a static structure.

Petri Nets defines the following basic concepts to define:

- The static structure
    * places (empty cirlce): represent the points of execution of a process.
    * transitions (labelled recangules): mark the current state of the process.
    * connections (archs): coonect places with transitions.
- The dynamic structure
    * tokens (black circle inside places): represent the current state of a system. The movement of tokens, according to the structure and the firing rules, is called token play.

---

9) The candidate is invited to briefly introduce the concept of LPAD.

[Possible question]

Probabilistic Logic Programming (PLP) defines a probability distribution over normal logic programs, called worlds. Joint distributions can also be defined over worlds to obtain the probability of a query.

Logic Programs with Annotated Disjunction (LPAD) is a language that extend a normal logic program by adding to the head of a clause some disjunctions of atoms, and each atom is annotated with a probability, i.e. each rule has a probability distribution over its head. For example:

sneezing(X):0.7 ; null:0.3 :- flu(X).

sneezing(X):0.8 ; null:0.2 :- hay_flue(X).

In LPAD, worlds will be obtained by selecting one atom from the head of every grounding of each clause. Moreover, given a clause C and a substitution θ such that Cθ is ground, we can define:

- Atomic choice (C, θ, i): is the selection of the i-th atom the the head of C for grounding Cθ.

- Composite choice k: is a set of atomic choices. The probability of a composite choise is $P(k) = \prod P(C,i)$ for each (C, θ, i) in k.

- Selection σ: is a total composite choice, i.e. there is one atomic choice for every grounding of each clause. A selection identifies a world w_σ and its probability would be $P(w_\sigma) = \prod P(C,i)$ for each (C, θ, i) in σ.

In LPAD, given a ground query Q and a world w, the probability of Q being true is 1 if Q is true in W, otherwise it is 0. The overall probability of a query is P(Q) = ∑ P(w) for each w such that w |= Q. For example, from:

```
sneezing(bob):0.7 ; null:0.3 :- flu(bob).

sneezing(bob):0.8 ; null:0.2 :- hay_flue(bob).
```

we can generate 4 different worlds. Given the query ":- sneezing(bob).", P(sneezing(bob))=P(w_1)+P(w_2)+P(w_3) since w_4 |= Q is false.

---

10) The candidate is invited to briefly introduce the concept of Ontology.

[Possible question]

An ontology is a formal (adopt a language with non-ambiguous semantic) and explicit (information must be derivable through a finite and sound procedure) description of a domain of interest.

In particular, an upper ontology is an ontology that focuses on most general domain.

Two properties can be defined for upper ontologies:

1. they should be applicable to almost any special domain.
2. combinign general concepts should not incur in inconsistencies.

Four approcaches are defined to create upper ontologies:

1. created by philosophers/logician/researchers.
2. created by automatically extracted knowledge from well-structured datasets.
3. created from text documents
4. created by using crowd-sharing information.

Categories and objects are the two main entities defined within ontologies because we can use these elements to reason upon a domain. In First-Order Logic (FOL), categories can be represented in two ways:

a. categories as predicates (e.g. Car(c1))

b. through reification, i.e. categories as objects (e.g. Member(c1, Car))

For reification, following properties can be defined:

- Membership (c ∈ Car): an object belongs to a category.
    * [necessity] members of a category enjoy some properties (e.g. (c ∈ Car) ⇒ hasWheels(c))
    * [sufficiency] members of a category can be recognized by some properties (e.g. hasWheels(c) ∧ hasPlate(c) ⇒ (c ∈ Car))
    * categories can enjoy properties as well (e.g. Car ∈ VehicleType)
- Subclass (Car ⊂ Vehicle): a category is a sub-category of another.

Also, some operations can be defined:

- Disjointness -> Given a set of categories S, the categories in S are disjoint iff they all have different objects:
    * Disjointness(S) ⇔ (∀c1,c2: c1 ∈ S ∧ c2 ∈ S ∧ c1 != c2 ⇒ c1 ∩ c2 = ∅)
    * e.g. Disjoint({Animals, Vegetables})
- Exhaustive Decomposition -> Given a category c and a set of categories S, S is an exhaustive decomposition of c iff any instance in c belongs to at least a category in S:
    * ExhaustiveDecomposition(S, c) ⇔ (∀i: i ∈ c ⇔ ∃c2: c2 ∈ S ∧ i ∈ c2)
    * e.g. ExhaustiveDecomposition({Male, Female}, People)
- Partition -> Given a category c and a set S of categories, S is a partition of c if:
    * Partition(S,c) ⇔ Disjoint(S) ∧ ExahustiveDecomposition(S, c)
    * e.g. Partition({Male, Female}, People)

Two types physical composition can be defined:

- Part-of -> if the objects have a structural relation
    * e.g. PartOf(engine, car)
- Bunch-of -> if the obects do not have a structural relation. It is useful to define a composition of countable objects.
    * e.g. BunchOf({bread1, bread2, bread3})

Objects can have measures among its types of properties:

- quantitative measures -> something that can be measured using some unit.
    * e.g. length(table1) = cm(80)
    * they propagate when using PartOf or BunchOf
- qualitative measures -> something that can be measured using terms with a partial or total order relation
    * e.g. {good, neutral, bad}
    * they do not propagate
    * fuzzy logic provides a semantic