FAIKR-Mod2

A common language: First Order Logic

We need to agree to a language that is free of ambiguity.

A notation for representing reasoning: symbols, the act of reasoning, interpretations

We have a "bootstrap" problem:

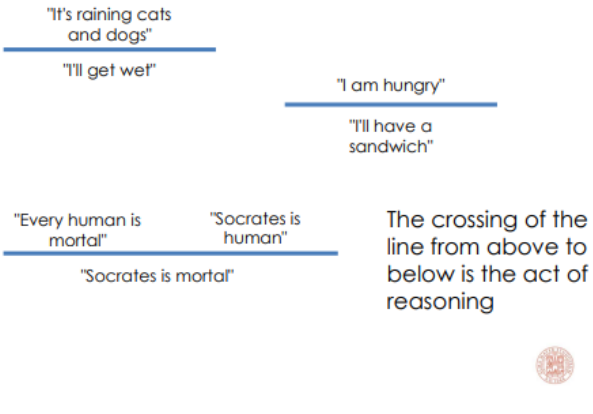
- we want to discuss on how to represent knowledge and reasoning
- better, we want to discuss "systems" for representing knowledge and do some (automatic) reasoning (and determine their properties, how they are made, etc.)… but we are missing any way for representing the object of our dialogues…

Solution: Let us use symbols

- Symbols can be anything, for us they will be signs drawn on a piece of paper or a slide
- We will associate a symbol to a concept (more on this, later)
- We will use symbols together and create sentences
- Sentences, in turn, might be true or false

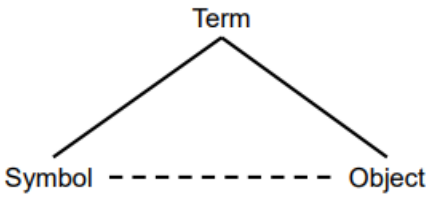
• We will represent the "act of reasoning" in the following way:

- we will write the things that we know (or believe we know, or that we know they are true)
- a horizontal line
- below the line, the result of our reasoning act



And the symbols?

• Historically, a long discussed topic. We will refer to the semiotic triangle by Ogden and Richards:



Example:

- *a symbol "pizza" made of five graphical traits.*
- *my personal idea of pizza.*
- *the pizza I will have tonight, tangible, physical, eatable, gorgeous, luxurious pizza.*

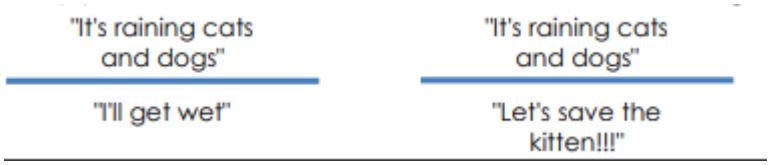
The interpretation of symbols

An immediate problem: who decides the links between symbols and ideas/objects?

- Everyone in this classroom has the right to decide the meaning of a symbol.
- Everyone in this classroom has its own, rightful interpretation of symbols.

- The truth of a sentence depends on the meaning of the symbols.

If anyone can have her/his own interpretation, what happens to the "correctness" of the reasoning?



Def : interpretation: An interpretation *I* is a pair (D, I) where:

- D is any set of objects, called domain
- I is a mapping called the interpretation mapping, from the (non logical) symbols to elements and relations over D

Given a set of sentences, their truth value will change depending on the interpretation we will give to the symbols: there is a strict relation between the notion of interpretation and the truth value.

Can we reasons, then?

The idea of Logical Consequence

To avoid the problem of the many different interpretations of symbols, we will restrict:

- from the notion of truth (something more related to philosophy)…
- … to the notion of Logical Consequence

Def: Logical Consequence: Let $\Gamma = F_1, .., F_n$ be a set of sentences (hypotheses or premises, and be a sentence (conclusion). F is a logical consequence of , written , if for any possible different interpretation, it is always true that if all the formulas in are true (in the world under examination) then also is true.

Propositional logic

Propositional logic is the simplest logic. It illustrates basic ideas using propositions Pi.

- Each Pi is referred as an **atom** or atomic formula
- Atoms Pi can be connected together by **connectives** – $\wedge$, $\vee$, $\rightarrow$, $\neg$, $\leftrightarrow$, $\perp$

• A well-formed formula is:

- an atomic formula Pi
- if A and B are well-formed formulas, then A $\wedge$ B, A $\vee$ B are well-formed formulas.

Each atom Pi can be either true or false.

Def: interpretation: Given a propositional formula G, let be $A_1, \ldots A_n$ the set of atoms which occur in the formula, an interpretation I of G is an assignment of truth values to $A_1, \ldots A_n$.

What about the non-atomic formula G? Interpretation is extended through the "truth tables" for the logical connectives.

Def: truth: Given an interpretation I, a formula G is said to be true in I if G is evaluated to true in the interpretation.

Given a formula G and an interpretation I, if G is true under I we say that I is a model for G, and usually write **I ⊨ G.**

There can be "extreme situations":

- no matter how we will choose an interpretation, G is always true: G is a **valid** formula, or a **tautology** (⊨ G)
- no matter how we will choose an interpretation, G is always false: G is an **inconsistent formula**
- mind: **invalid is different from inconsistent**. More frequently, there will be formulas G that are invalid, bur are satisfiable (consistent).

Propositional Logic and Decidability

Propositional Logic is decidable: there is always a terminating method to decide whether a formula is valid.

How to decide if a formula is valid or not?

1. enumerate all the possible interpretations
2. look at the formula for each interpretation Which (computational) cost?

Example: a small problem of logic network for controlling a railway station with one train track, can be represented with 30000 propositional atoms. Determining properties about such a network would consist to explore up to 2^30000 different interpretations.

Propositional Logic is decidable but this does not help us to efficiently decide.

WAIT! which is our goal?

- Goal: to decide if G is a valid formula. But we can do better:
- Goal: to decide if, given some formulas are true, a formula G is true as well

It is a reformulation of the goal above: we could always substitute each true atom with true, simplify G, and check if G is valid for all the possible interpretations. This perspective allows us to introduce the notion of **logical consequence.**

Def : Logical Consequence: Given a set of formulas $F_1, .., F_n$ and a formula G, G is said to be a logical consequence of $F_1, .., F_n$ if for any interpretation I in which $F_1 \wedge .. \wedge F_n$ is true, G is also true. Syntactically: $F_1 \wedge .. \wedge F_n \vDash G$

Def: Logical Equivalence: Def.: Two formulas F and G are logically equivalent F ≡ G if the truth values of F and G are the same under every interpretation of F and G. F ≡ G if $F \vDash Q, Q \vDash G$.

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

Reasoning

The notion of logical consequence allows us to "forget" the problem of the many different interpretations.

Is there a way that allows us to deal with the formulas, and to decide if a specific formula G is valid?

Many different methods for working with the symbols have been identified. Working with the symbols is indeed indicated as reasoning.

- usually we refer to a reasoning engine/method, and use the syntax$F_1 \wedge .. \wedge F_n \vdash^{\mathrm{E}} \mathrm{G}$ to denote that G can be deduced by $F_1 \wedge .. \wedge F_n$using the reasoning method E

In general, we will write $KB \vdash^{E} G$ where KB is a knowledge base in some form/language/notation/logic.

Theorem: Reasoning by deduction: Given a set of formulas $F_1 \wedge .. \wedge F_n$ and a formula G, $F_1 \wedge .. \wedge F_n \vDash G$ if and only if $\vDash (F_1 \wedge .. \wedge F_n) -> G$

How to use it for reasoning?

We can prove the logical consequence of G by proving the validity of ._$\vDash (F_1 \wedge .. \wedge F_n) -> G$.

Theorem: Reasoning by refutation: Given a set of formulas and a formula G, $F_1 \wedge .. \wedge F_n \vDash G$ if and only if $F_1 \wedge .. \wedge F_n \wedge \neg G$ is **inconsistent.**

How to use it for reasoning?

We can prove the logical consequence of G by proving the inconsistency of ._$F_1 \wedge .. \wedge F_n \wedge \neg G$.

Reasoning by refutation yes but how to? The idea of using refutation for reasoning was presented in:

Robinson , J. Alan (1965): **A Machine-Oriented Logic Based on the Resolution Principle**, Journal of ACM, 12 (1): 23–41.

To do so, the author proposed to exploit the Resolutionrule, together with the refutation mechanism.

Resolution rule applies to logic formulas expressed in the form of clauses.

A clause is a disjunction of literals, i.e. atoms and negated atoms: $A_1 \vee, \ldots, \vee \mathrm{A_n} \neg \mathrm{B_1} \vee \ldots, \vee \neg \mathrm{B_n}$.

• Let us suppose to have two clauses such that:

- $C1 = A_1 \ldots \vee \mathrm{A_i} \ldots, \vee A_n$
- $C2 = B_1 \ldots \vee \neg A_i \ldots, \vee \mathrm{B_n}$

Then the following clause:

- C=$A_1 \ldots \vee \ldots, \vee A_n \ldots B_1 \vee \ldots, \vee B_n$ is a logical consequence of C1 and C2.

Resolution $\dfrac{(A \vee B) \qquad (\neg B \vee C)}{(A \vee C)}$

A	B	C	(A ∨ B)	(¬B ∨ C)	(A ∨ C)
F	F	F	F	T	F
F	F	T	F	T	T
F	T	F	T	F	F
F	T	T	T	T	T
T	F	F	T	T	T
T	F	T	T	T	T
T	T	F	T	F	T
T	T	T	T	T	T

Remember we are interested in **logical consequence!**

Summary:

- Goal: we want to know if $\underline{F_1 \wedge .. \wedge F_n \models G}$
- From theorem, we know that we can prove the above if we prove . $\underline{F_1 \wedge .. \wedge F_n \wedge \neg G}$ is inconsistent
- Change our goal: prove that . $\underline{F_1 \wedge .. \wedge F_n \wedge \neg G}$ is inconsistent, i.e. there is no possible interpretation I that makes true the formula.
- How? through resolution, we derive the inconsistency.

Exercise:

Exercise:
Given the following
$KB = \{(a \rightarrow c \vee d) \wedge (a \vee d \vee e) \wedge (a \rightarrow \neg c)\}$
Prove that $G = \{(d \vee e)\}$ is logical consequence of KB.

Solution:
i. Transform in CNF the KB_{CNF}:
1. $(\neg a \vee c \vee d)$
2. $(a \vee d \vee e)$
3. $(\neg a \vee \neg c)$

ii. Negate the formula G:
4. $\neg d$
5. $\neg e$

iii. Derive the possible resolvents:
First iteration:
6. $(1+2)$: $(c \vee d \vee e)$
7. $(2+3)$: $(\neg c \vee d \vee e)$
8. $(1+4)$: $(\neg a \vee c)$
9. $(2+4)$: $(a \vee e)$
10. $(2+5)$: $(a \vee d)$
11. $(1+3)$: $(\neg a \vee d)$

Second iteration:
12. $(10+11)$: d

Third iteration:
13. $(4+12)$: empty clause -> contradiction -> the original formula G is logical consequence of the KB.

Refutation and Resolution: properties for the propositional setting

Sound? If $KB \cup \{\neg G\} \vdash^{RR} \perp$ then $KB \models G$

Complete? If $KB \models G$ then $KB \cup \{\neg G\} \vdash^{RR} \perp$

Decidable.

Efficient? not really, np-Complete in the worst case.

Expressive power? poor, unsatisfactory

Things are better if we restrict to Horn Clauses (at most one positive literal)

And we adopt the SLD rule (Linear resolution with Selection function for Definite clauses).

First order logic (FOL)

Five different sets of elements

1. The set of the constants (e.g., "federico")
2. The set of function symbols (e.g., son_of(federico))
3. The set of variables (e.g., X)
4. The set of predicate symbols
5. The set of logical connectives

- ¬ (negation),
- ∧ (conjunction),
- ∨ (disjunction),
- → (implication),
- ↔(equivalence)
- brackets "(" ")"
- Existential ∃ and universal ∀ quantificators.

Def Term: A term is recursively defined as

1. A constant;
2. A variable;
3. If is a n-ary function symbol, and $t_1, \ldots, t_n$ are terms, then f($t_1, \ldots, t_n$) is also a term.

Terms are introduced and used as symbols to indicate the objects of our discourse

E.g.: • federico • francesco • son_of(federico) === francesco

Def Atom: An atom/atomic formula is defined as the application of a n-array predicate symbol p to n terms $p(t\_1, \dots, t\_n)$

Predicates are introduced and used to reason about the truth of sentences.

E.g.: • awesome(federico) • teenager(francesco) • loves(federico, francesco).

First Order Logic: Well Formed Formulas

Well Formed Formulas (WFF) are recursively defined as:

- Every atom is a WFF
- If φ and ψ are WFF, then the following are WFF:
  - $\neg\varphi$
  - $\varphi \wedge \psi$
  - $\varphi \vee \psi$
  - $\varphi \rightarrow \psi$
  - $\varphi \leftrightarrow \psi$
- if φ is a WFF and X is a variable, then the following are WFF:
  - $\exists X\, \varphi$
  - $\forall X\, \varphi$

- A Literal is an defined to be an tom or its negation.

Upper ontologies and other things

Upper Ontologies

Def: Ontology: An ontology is a formal, explicit description of a domain of interest.

- **formal**: it should be described using a language with clear and non-ambiguous semantics
- **explicit**: the information should be readily available or, in the worst case, derivable in a finite time with a sound procedure
- **description**: it should provide us information… more interesting, the information it provides define the goal of the ontology itself
- **domain**: such description should be related to some topic of interest, and in turn it should be related to the ontology goal.

We (humans) continuously make use of ontologies. It seems that is an evolutionary trait of our brain, and an economic way for

- storing information
- communicating information

Generally speaking, we (humans) tend to organize our knowledge through two notions:

- Categories
- Objects (belonging to the categories)

Example: university employers

Problem: the new university director wants to organize all the employees, with the only purpose of the salary management. University employees belong to:

- *the administratives group*
- *the technicians group;*
- *the researchers/teachers group*
- *administrative and technical staff are organized into four different groups (B, C, D, EP, each with a different salary)*
- *researchers are organized in many different groups (because we are in Italy); roughly speaking: researchers, associates and full professors.*

Few questions:

- *Does it make sense to keep the distinction between administrative and technical staff?*
- *if the focus is the salary only… may be no*
- *Is it possible that a person belongs at the same time to two categories?*

Each university employee has a fiscal code (for taxation).

- *• To be precise, every resident in Italy has a fiscal code.*
- *This could suggest that there is a further category, the one of "resident in Italy"*
- *In turn, there could be a more general category of "Person"*
- *What is a "Person"? WRONG QUESTION: we started from the salary, let's focus on the salary…*

Example: travel means

Citizen of Bologna use several different travel means. The major wants to put some order and evaluate some new green policy.

- *Travel means can be divided into: public vs. private transports.*
- *Travel means can be divided into:*
    - *– those that fly*
    - *those that sail*
    - *those that run on railways*
    - *those that run on wheels on concrete and asphalt*

Apparently, there are two orthogonal dimensions • Automobiles belongs to a category • My car is an instance of one category.

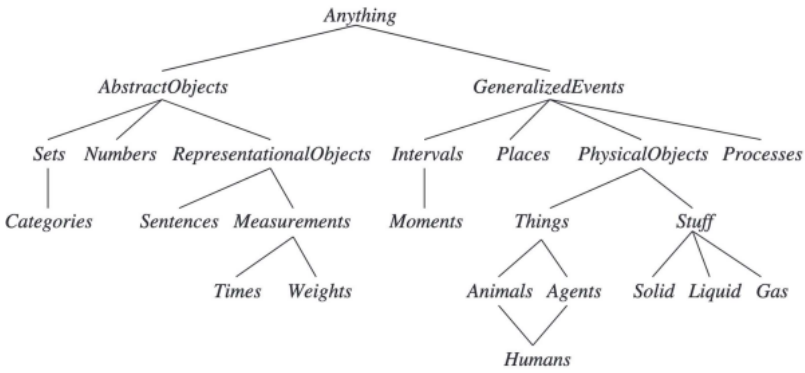
Categories, hierarchies, and objects

- There are categories (staff vs. professors)
- Categories can be more or less general (person vs. professor)
- Categories can be organized hierarchically
- Objects can be instances of categories
- Objects can belong to more categories at the same time

Upper ontologies

Among the categories, there can be:

- very general categories (e.g., Person)
- very specific ones (e.g., Employee of the University of Bologna in the administrative part, category D)



Generally speaking, this distinction between "very general" and "very specific" can be extended to the domain we want to describe, and then to the ontology that describes it.

Def :Upper ontology: An upper ontology is an ontology that focuses on the largest, more general, more high level, most general purpose domain.

How special purpose ontologies relate with general purpose ontologies?

Starting from special purpose, you can always move up (through generalisation), towards more general concepts "Do all these ontologies converge on a general purpose ontology?" "Maybe" [cit. AIMA]

At least two characteristics of upper ontologies:

1. they should be applicable to (almost) any special domain
2. we should be able to combine different general concepts without incurring in inconsistencies (e.g., time and space concepts should work together…)

Categories

• It is commonly accepted that humans represent the knowledge in terms of categories and objects belonging to: However, this alone would not be a strong/sufficient motivation for introducing categories

Part of the (human) reasoning happens at the level of categories

- Our goals can be specific-instance-driven *E.g., I want to have dinner with my wife, I want to fuel my car, I want to*
- Our goals can be category-driven *E.g., I want to buy tomatoes, I'd like to read a (copy of) a book with title "AIMA.*

We make **predictions** about the (future) evolution of the world using categories. We recognize objects as belonging to certain categories. We assign properties to the (objects belonging to) categories. We assign models to the (objects belonging to) categories. We then make "predictions".

E.g.: I am able to drive "any" car; I am able to use any fork and knife (at restaurant).

How to represent them?

Objects: it is safe to assume there exists always a way to identify each of them, a unique name.

E.g.: My car has a plate, I have a fiscal code, every pc on the internet has an IP address. Notice: this is not always true, there is part of the world that defies this assumption.

In FOL, two possible alternatives:

1. categories as predicates *e.g.: Car is a predicate that represents the category of cars; Car(aa123bb) means that the vehicle with plate aa123bb is a car*

   How to represent that (every) Car is a special type of (every) Vehicle? *Sub(Car, Vehicle) … Not practical! it's a second order logic.*

2. through reification: categories as objects as well *E.g.: Member(aa123bb, Car) or aa123bb appartiene Car.*

What we would like to express about categories and objects? Is reification good enough?

- Membership
- Subclass: categories are organized in hierarchies, and subcategories inherit properties of the parents Note that the subclass relation generates taxonomies.

What we would like to express about categories and objects?

- **Membership.** *Aa123bb appartiene a Car*
- **Subclass.** *Car sottoinsieme Vechicle*
- members of a category enjoy some property (necessity).
- members of a category can be recognized by some properties (sufficiency).
- categories themselves can enjoy properties.

Disjointness

Categories relate each other because of subclass relation…what else?

E.g.: Linneus stated that every animal could not be a vegetable, and the opposite. We would say that Animals and Vegetables are two disjointed categories, although they are subclass of LivingThing.

Formally:

$$Disjoint(s) \Leftrightarrow (\forall c_1, c_2 \quad c_1 \in s \ \wedge \ c_2 \in s \ \wedge \ c_1 \neq c_2 \ \Rightarrow \ c_1 \cap c_2 = \emptyset)$$

E.g.: Disjoint({Animals, Vegetables}).

Exaustive Decomposition

Subcategories might "cover" all the possible instances of the parent category.

E.g.: Every one in this classroom is a student, or a professor (or both). "Student" and "Professor" categories provide an exhaustive decomposition of the category "People in this classroom"

Formally, given a category c, and a set s of categories, s is an exhaustive decomposition of c if :

$$ExhaustiveDecomposition(s, c) \Leftrightarrow (\forall i \quad i \in c \ \Leftrightarrow \ \exists c_2 \quad c_2 \in s \ \wedge \ i \in c_2)$$

E.g.: ExhaustiveDecomposition({Student, Professor}, PeopleInThisRoom).

Partition

If a category can be decomposed in more categories such that:

- they form an exhaustive decomposition
- they are disjoint

Then we have a partition.

E.g.: Linneus stated that every living thing would partitioned into three subcategories (nowadays five categories).

Formally, given a category c, and a set s of categories, s is a partition of c if :

$$Partition(s, c)$$
$$\Leftrightarrow$$
$$Disjoint(s) \wedge ExahustiveDecomposition(s, c)$$

Physical Composition

Objects can be made of other objects… the relation between an object and its parts is called meronymy: some objects (**meronyms**) are part of a whole (**holonym**).

E.g:

- *A car is composed of an engine, four wheels, five seats,…*
- *A laptop is made of a motherboard, a cpu, a keyboard, a screen, etc.*
- *My shopping bag contains three pieces of bread*

The (philosophical) discipline that focuses on parts, on the relations among them, and relations with the whole, is named **mereology.**

A rough, quite general distinction in the physical composition stems from the answer to this question: is there any **structural relation** between the parts, and between the parts and the whole? (other than the composition itself?).

- Answer yes: the relation is usually named **part of**. *E.g.: PartOf(engine123,car_aa123bc), PartOf(cylinder1345, engine123)*
- Answer no: the relation is usually named **BunchOf.** *E.g.: BunchOf({Bread1, Bread2, Bread3})*

Part of

The PartOf relation enjoys some properties:

- Transitivity $PartOf(x, y) \wedge PartOf(y, z) \Rightarrow PartOf(x, z)$
- Reflexivity $PartOf(x, x)$

always holds.

BunchOf

The BunchOf relation aims to define objects in terms of composition of other countable objects…

- … however, a structure in the composition relation is missing. *E. g.: boxOfNails17 = BunchOf({nail1, nail2, nail3, nail7}).*

Why not use Set?

Set is a mathematical concept.Properties in general cannot be propagated to the Set concept.

E.g.: the weight of the box of nails is given by the sum of the nails contained in the box… the weight property of each nail has a relation with the weight of the box.

Measures

Among the many types of properties, objects can have measures, and specifically:

- **quantitative measures**: something that can be measured in a quantitative manner, w.r.t. some unit. *E.g.: the duration of this lesson, measured in seconds; the dimension in the space of a table (W x L x H).*
- usually represented using a unit function that takes in input a number *E.g.: length(table1) = cm(80).*
- properties expressed in quantitative measures can/usually propagate upwards/downwards in the PartOf/BunchOf relations. *E.g.: the total weight of a car is given by the sum of the weights of its parts.*
- **qualitative measures**: something that can be measured using terms, and these terms come with an total/partial order relation. *E.g.: this lessons is {devastatingly boring, just boring, puzzling, amusing, fantastic} on a boring scale: implicitly: [dev.boring < boring < puzzling < amusing < fantastic].*
- they do not propagate downward/upward. *E.g.: two boring lessons in the same day do not make the day twice boring, or devastatingly boring, or boring2…*
- the presence of an order relation however allow their use in object/category definitions.

Qualitative Measures and Fuzzy Logic

What if we really need to make quantitative reasoning over qualitative measures?

Fuzzy Logic provides a (many-values-truth) semantics to qualitative measures

- many semantics, Lukasiewicz's one being among the more mentioned
- semantics focuses on the meaning of standard logical connectives (AND, OR, NOT)…
- … and on the combination of rules

Why not a probabilistic approach? Probabilistic logic and fuzzy logic capture different aspects of human reasoning…

Things or Stuffs?

Till here, categories and objects were about things that can be individuated, or distinct from each other.

- *we can spot the car of the professor*
- *we can spot a single apple in the bunch of apples*
- *we can spot the people in this room, each one is an individual*

Whether explicitly or implicitly:

- there is some **individuation criteria**
- mainly referred to the PartOf/BunchOf relations
- mainly related to the substance of the object itself

- related with the logical minimization criteria

There are objects in our real world that defy the individuation criteria. In English:

- count nouns: car, apple, human
- mass nouns: water, air, butter

In the Cyc ontology, and Russel&Norvig as well, they adopt the distinction between:

- **Things**
- **Stuff**

Beware: a glass of water is different from water…

A solution comes when looking at the properties

- some properties are referred to the substance of the object *E.g.: the water boils at 100°C*
- Such properties are named **intrinsic**, and they are retained even when some division is applied
- Some properties refer to the object, and to the object structure relation with its parts *E.g.: the weight of professor's car change if we take out a wheel*

Such properties are named **extrinsic.**

Extrinsic and intrinsic properties

A glass of water:

- has a volume
- has a weight
- is made of two "parts": the glass itself, and the "water within", each one with a different weight and volume (???)
- if we divide a glass of water, we will not obtain two glass of waters

The water:

- We can (infinitely?) divided it, and still it will be water.
- It boils at a certain temp.
- We can split it, and it will boil at the same temp.

Which one to use, and when?

A solution comes when looking at the properties:

- a category of objects that will include in its definition only intrinsic properties is a substance; **Stuff** is the most general substance category
- a category of objects that will include in its definition any extrinsic property is a count noun; **Thingis** the most general object category.

(Qualitative) Reasoning over Time

Propositional Logic for representing dynamics, and the Frame problem

Wittgenstein in a 1922 essay faced the problem of representing the belief of an agent w.r.t. the world.

The idea:

- use propositional logic, and **propositions**, to represent an agent's beliefs about the world
- **a current state of the world would be represented as a set of propositions**
    - in the set it would appear only "true" propositions, thus representing what the agent believes to be true
- Different sets, put in some order, would represent the world evolution perceived by an agent
    - sets and propositions in the sets would be marked with numerical apex to establish the evolution order

Example:

*A child has a bow and an arrow. She can shoot the arrow, throw the bow, but she can also run, hide, and crouch. Our agent **observes** the child, and it believes/knows that:* KB^0 = {hasBow^0, has Arrow^0}

*The child shoots the arrow, our agent **observes** the child and it believes/knows that: KB^1 = {hasBow^0, has Arrow^0, hasBow^1, ¬hasArrow^1}.*

We are interested to focus on the dynamic, or rather how the world evolves from KB^0 to KB^1

- What happened in between?
- There is some idea of action, whose execution changes the world…

How to represent the dynamic of the world?

Def: State: Notion that captures the world at a certain instant.

The evolution of the world is given as a sequence of states…

States are **countable** …

There are **actions** …

Def: Actions: Actions affect how each state evolves to the next one…We would describe such effect in terms of **effect axioms.**

Effect Axioms

Example.

A child has a bow and an arrow. She can shoot the arrow, throw the bow, but she can also run, hide, and crouch. The child shoots the arrow…

What does it mean "shooting an arrow"?

$\text{shoot}^t \Longrightarrow (\text{hasArrow}^t \Leftrightarrow \neg\text{hasArrow}^{t+1})$

$KB^0 = hasArrow^0$

$KB^1 = \text{hasArrow}^0, \neg\text{hasArrow}^1$

WAIT! That is different from:

$\mathrm{KB}^1 = \mathrm{hasBow}^0, \mathrm{hasArrow}^0, \mathrm{hasBow}^1, \neg\mathrm{hasArrow}^1$

Does the child still have the bow after the shoot? In other word, hasBow1 is TRUE or FALSE?

The Frame problem

- The effect axioms fail to state what remains unchanged as the result of an action.
- It is named **"Frame problem"** because it refers to the fact that the "background" remains unchanged, while the "foreground" is subjected to the effects…
- … but the "effect axiom" tell us something about the "foreground"... what about the "background", alias frame?

A solution would be the **frame axioms**. For each proposition that is not affected by the action, we will state that it is unaffected.

Problems:

- If we have m actions and n propositions, the set of frame axioms will be O(mn)… representational frame problem.
- Reasoning about t steps ahead will have a temporal complexity of O(nt)… inferential frame problem.
- Locality principle mitigate the problems above, does not eliminate them.

Objection: do we really need to assert frame axioms?

- we could use logics for stating a "sort of super axiom" that say that everything that is untouched remain the same.
- it is not possible to do it in FOL, we can do it in high order logic.

The Situation Calculus

The Successor State Axioms

Effects and Frames axioms focuses on the actions .A solution consists on changing the viewpoint: focus on the propositions describing the world, that we will be named **fluents.**

Theorem: Successor State Axioms: Each state is described by a set of fluents F. Then, we define the following axioms: $F(t+1) \Leftrightarrow ActionCausesF^t \vee (\mathrm{F}^t \wedge \neg\mathrm{ActionCausedNotF}^t)$

Example: $\mathrm{hasBow}^{t+1} \Leftrightarrow \mathrm{pickUpBow}^t \vee (\mathrm{hasBow}^t \wedge \neg\mathrm{throwBow}^t)$

Proposed by John McCarthy (1963); Ray Reiter (1991).

It aims at Planning as first-order logical deduction

- The initial state is called a **situation**
- If a is an action and s a situation, then **Result(s, a) is a situation**
- A function/relation that can vary from one situation to the next is called a **fluent**: At(x, l, s)
- Introduces **preconditions** of an action (as in planning)
- Action's preconditions are defined by possibility axioms: $\phi(\mathrm{s}) \Longrightarrow \mathrm{Poss}(\mathrm{a}, \mathrm{s})$

E.g.: hasArrow(s) ∧ hasBow(s) ⟹ Poss(shoot, s)

Adopt the Successor state axioms, but adapted with the possibility notion:

F^ (t+1) ⟺ ActionCausesF^t ⋁ (F^t ⋀ ¬ActionCausesNotF^t).

Poss(a, s) ⟹ (F(Result(a,s)) ⟺ a=ActionCausesF ⋁ (F(s) ⋀ a ≠ ActionCausesNotF).

Poss(shoot, s) ⟹ (¬hasArrow(Result(a,s)) ⟺ a=shoot ⋁ ¬hasArrow(s) ⋀ a ≠ reload).

- To avoid non-determinism and/or conflicts/incoherencies, it has a **"unique action" axiom**: only one action can be executed in a situation.
- The goal is defined as a conjunction of fluents.
- The solution is a situation, i.e. a sequence of actions, that satisfies the goal.
- Deduction is used to compute the right action sequence.

McCarthy's and Green's formulation maps fluents into predicates; they do not allow to assert about the truthness of a fluent in a situation. Actions are discrete, instantaneous, and happen one at a time. Two actions cannot happen at the same time.

The Situation Calculus defines:

- What is true before the action…
- What is true after the action…
- … but say nothing on what is true during the action

The Event Calculus

Proposed by Marek Sergot and Robert Kowalski, 1986

- Based on points of time
- Reifies both fluents and events into terms: HoldsAt(fluent, situation/action).
- **Fluents** are properties whose truthness value changes over time • Advantages of a second-order formula, yet still first order
- Allows to link multiple different events to the same state property (named fluent)
- State property changes can depend also from other states.
- Allows to reason on meta-events of state property changes (clip and de-clip meta-events)

The formulation comprises an ontology and two distinct set of axioms:

1. **Event calculus "ontology"** (fixed)
2. **Domain-independent axioms** (fixed)
3. **Domain-dependent axioms** (application dependent)

Event Calculus : Ontology.

- **HoldsAt(F, T):** The fluent F holds at time T.
- **Happens(E, T):** event E (i.e., the fact that an action has been executed) happened at time T.
- **Initiates(E, F, T):** event E causes fluent F to hold at time T (used in domain-dependent axioms…)
- **Terminates(E, F, T):** event E causes fluent F to cease to hold at time T (used in domain-dependent axioms…).
- **Clipped(T1, F, T):** Fluent F has been made false between T1 and T(used in domain-independent axioms), T1< T.
- **Initially(F)** : fluent F holds at time 0.

Event Calculus: Domain-independent Axioms.

Two axioms define when a fluent is true:

- HoldsAt(F, T) ⟸ Happens(E, T1) ⋀ Initiates(E, F, T1) ⋀ (T1< T) ⋀ ¬Clipped(T1, F, T)
- HoldsAt(F, T) ⟸ Initially(F) ⋀ ¬Clipped(0, F, T)

An axiom defines the clipping of a fluent:

- Clipped(T1, F, T2) ⟸ Happens(E, T) ⋀ (T1< T < T2) ⋀ Terminates(E, F, T)

Event Calculus: Domain-dependent Axioms

A collection of axioms of type Initiates(…)/Terminates(…), and Initially(…).

- **Initially(F):** the fluent F holds at the beginning.
- **Initiates(Ev, F, T):** the happening of event Ev at time T makes F to hold; it can be extended with many (pre-)conditions.
- **Terminates(Ev, F, T):** the happening of event Ev at time T makes F to not hold anymore.

Event Calculus: Example

Example: we have a single button in a room: the pressing of the button switch on or off the light.

Fluents:

- Light_on, light_off why not light(on) vs light(off) ??? it's fine as well…

Events:

- push_button.

Event Calculus : Example

Example: we have a single button in a room: the pressing of the button switch on or off the light.

Domain-dependent axioms, initial state of the world:

- Initially(light_off).

Effects of the "push_button" event on the fluent **light_on**:

- Initiates(push_button, light_on, T) ⟸ HoldsAt(light_off, T).
- Terminates(push_button, light_on, T) ⟸ HoldsAt(light_on, T)

Effects of the "push_button" event on the fluent **light_off**:

- Initiates(push_button, light_off, T) ⟸ HoldsAt(light_on, T).
- Terminates(push_button, light_off, T) ⟸ HoldsAt(light_off, T).

Event Calculus : Example

Given a set of events:

- Happens(push_button, 3)
- Happens(push_button,5)
- Happens(push_button, 7)
- Happens(push_button, 8)

Is the light on?

Event Calculus allows to answer the queries about HoldsAt predicates very easily.

Event Calculus is very important… why?

It allows to represent the state of a system in logical terms, in particular FOL. It allows to represent and reason on how a system **evolves** as a consequence of happening events. It is an easier formalism (w.r.t. other solutions). It is declarative/logic based.

Applications:

- Monitoring of systems.
- Simulation of system's evolutions.
- Practically, adopted as a principled framework for representing system evolutions and reactiveness.

Event Calculus :few limits…

- Easily implemented in Prolog…
- …but (roughly) not safe if fluents/events contain variables, due to the negation in front of the clipping test, that is implemented in Prolog through NAF.

Event Calculus : few limits…

Allows deductive reasoning only:

- Takes as input the domain-dependent axioms and the set of happened events…
- Provides as output the set of fluents that are true after all the specified events.
- What if a new happened event is observed?
- New query is needed: re-computes from scratch the results… computationally very costly!!!!

Extensions: Reactive Event Calculus

- Overcome the deductive nature of the original formulation given by Sergot & Kowalski
- New happened events can be added dynamically, i.e. the result is updated (and not re-computed from scratch)
- Allows events in a wrong order
- More efficient
- Can be implemented in backward reasoning as well as in forward reasoning

Allen's Logic for reasoning over temporal aspects

EC is based on the notion of events or, better saying, on the happening of things.

- Are these events instantaneous? Or do they have a duration?

Before, how do we measure the passing of time?

- Every time scale is an arbitrary reference system.
- We need an origin *E.g.: 1 January 1970*.
- We can choose a measurement unit *E.g.: seconds*
- Points in time are measured w.r.t. distance from the origin.

The first question is not trivial: there are pro&cons for both the choices. However, there is a consensus that duration-based representation of events is richer.

Notice:

- a durative event can be represented in terms of start and end time points
- an instantaneous event has duration zero

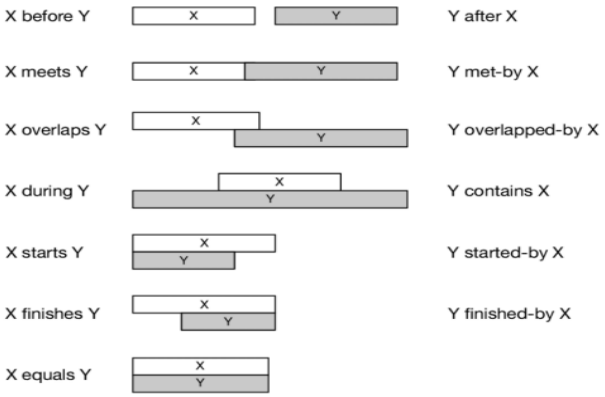
Allen's logic of intervals

Def: Begin(i): an interval i begins at a certain time point: we introduce the function **Begin(i**) that return that timepoint.

Def:Ends(i): an interval i ends at a certain time point: **Ends(i).**

Allen proposed 13 temporal operators:

- Meet(i,j) ⇔ End(i) = Begin(j)
- Before(i, j) ⇔ End(i) < Begin(j)
- After(j, i) ⇔ Before(i, j)
- During (i, j) ⇔ Begin(j) < Begin(i) < End(i) < End(j)
- Overlap(i, j) ⇔Begin(i) < Begin(j) < End(i) < End(j)
- Starts(i, j) ⇔ Begin(i) = Begin(j)
- Finishes(i, j) ⇔ End(i) = End(j)
- Equals(i, j) ⇔ Begin(i) = Begin(j) AND End(i) = End(j)



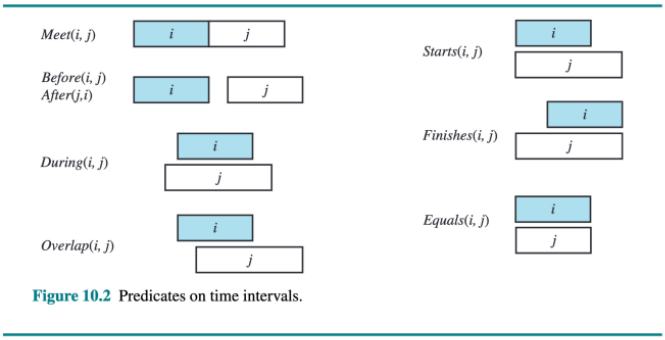


Figure 10.2 Predicates on time intervals.

Knowledge and Beliefs

Our agent in the environment

So far, we discussed about representing knowledge.

Implicitly, we adopted this point of view:

- our agent/program/decision system acts in a certain environment
- knowledge about the environment is available
- our agent take decisions/plan on the basis of the knowledge

The viewpoint of an agent with KB has some limits.

In the environment there are other agents:

- Success of any decision depends on the actions taken by other agents.
- These others agents might have their own knowledge…
- … knowing something about the others might help to take our own decisions?

Our agent has a KB, but it does not know it:

- no distinction between knowledge, belief, perception, …
- no possibility of any meta-reasoning on its own knowledge

Propositional attitudes

Examples:

- *There is a pizza in the fridge*
- *Federico knows that there is a pizza in the fridge*
- *Federico believes that there is a pizza in the fridge*
- *Federico wants a pizza*
- *Federico informs Elena there is a pizza in the fridge*
- *Elena knows there is a pizza in the fridge*
- *Federico knows that Elena knows there is a pizza in the fridge*
- *Federico informs Elena that he wants a pizza*
- *Elena knows Federico wants a pizza*
- *Federico knows Elena knows Federico wants a pizza*

The examples show what have been named propositional attitudes of an agent towards mental objects (its own, and of other agents). E.g.: Believes, Knows, Wants, Informs

How to represent propositional attitudes? Using FOL predicates?

- They refer to something that is not a term Knows(Federico, location(pizza, fridge)) location(pizza, fridge) is usually a sentence, not a term.
- FOL predicates might support referential transparency Knows(Lois, CanFly(Superman)) Superman=Clark Does Lois knows that Clark can fly? Propositional attitudes behave differently from predicates
- Using FOL predicates is not the best choice ("does not work").

Modal operators

The idea was to introduce new, special operators, named Modal operators, that behaves differently from logical operators.

Modal logic

Characteristics:

- same syntax as FOL
- Augmented with modal operators
- Modal operators take as input sentences (rather than regular terms)
- Each modal operator takes two input: the name of an agent, and the sentence it refers to
- Semantics is extended with the notion of possible worlds
- Possible worlds are related through an accessibility relation (that might depend on the operator)
- Different operators define different logics

Modal operators and Modal Logics

Example: operator K for indicating that agent a knows P:

$$\boldsymbol{K_a}\mathrm{P}$$

- Federico knows there is a pizza in the fridge Kfedericolocation(pizza,fridge)
- Federico knows that Elena knows that there is a pizza in the fridge $\mathrm{K_{federico}K_{elena}}$location(pizza,fridge).

Modal logics

The starting point: **an agent in a given scenario is not omniscient** regarding all the aspects of the current state of the world (the current world).

Thus, an agent might think:

- there is the current perception of the state of the world
- the unknown is depicted as possible worlds, with an intuitive notion of possibility or accessibility from my current world.

We are not really interested on a specific notion of "accessibility": it might suffice that there is such a notion. The idea is something like: an agent knows p if in any accessible world accessible from the current, p is true.

Semantics of Modal Logics: formally

Def: Kripke: Given a set of primitive propositions ϕ, a Kripke structure M=$(S, \pi, K_1, .., K_n)$ is defined :

- S is the set of states of the world
- π: $\phi \rightarrow 2^S$ specifies for each primitive proposition the set of states at which the proposition hold.
- $K_1, .., K_n$ are binary relations over S, i.e. $K_i \subseteq$ S ×S, with the meaning: (s,t) ∈ K_i" if agent i consider the world t possible from s.

Don't get confused by the accessibility/possibility relation: we are not really interested in it.

Example

Alice is in a room, and will toss a coin, the coin will land head or tail. Bob is in another room, he will hear the sound of the coin on the floor, will enter the room and observe if the coin is head or tail.

$M = (S, \pi, Ka, Kb)$:

- $\phi = (tossed, head, tail)$
- *Three stages: before Alice toss the coin, after Alice toss the coin, after Bob enters the room:* $S = (s_0 h_1, t_1, h_2, t_2)$, *where* s_0 *is the initial stage;* h_1, t_1 *capture that the coin has been tossed and landed head/tail;* h_2, t_2 *capture that Bob entered the room and observed the coin.*
- *Three stages: before Alice toss the coin, after Alice toss the coin, after Bob enters the room:*$S = (s_0, h_1, t_1, h_2, t_2)$.

Which propositions can be true in any stage?

- $\pi(tossed) = (h_1, t_1, h_2, t_2)$
- $\pi(head) = (h_1, h_2)$
- $\pi(tail) = (t_1, t_2)$

Alice in the room: from her viewpoint, she always perfectly observes each world, and there is no "possible worlds" left out:

- $K\_a = \{(s, s) \mid s \in S\}$

Bob observes everything in the first and the third stage, but is unsure in the second stage:

- $K_b = \{(s, s) | s \in S\} \cup \{(h_1, t_1), (t_1, h_1)\}$

Finally, we can establish the truth of sentences like:

$$(M, s_0) \models K_A \neg tossed \wedge K_B K_A K_B (\neg head \wedge \neg tail)$$

$$M, s_1) \models (head \vee tail) \wedge \neg K_B head \wedge \neg K_B tail \wedge K_B (K_A head \vee K_A tail)$$

Properties of the knowledge modal operator

Which are the axioms of our logic?

0) **Axiom A0:** All instances of propositional tautologies are valid .
   MP Modus Ponens: if ϕ is valid and $\phi \Rightarrow \Phi$ is valid, then Φ is valid.

What about the knowledge operator K w.r.t. the modus ponens?

1) **Axiom A1**: **Distribution Axiom**; $K\ i\ \phi \vee K\ i(\phi\ \rightarrow \Phi))\ \rightarrow K{:}\, i\ \Phi.$ in other words, knowledge is closed under implication.

The definition so far of knowledge assumes quite powerful agents…

Notice:

- if ϕ is true in all the worlds of a structure M (i.e., ϕ is valid)…
- … then ϕ must be true, in particular, in all the worlds (M,s) that agent i considers as possible.
- If ϕ is true in any possible world, then agent i knows ϕ (by definition) in all the possible worlds.

Def: (Knowledge Generalization Rule): G: for all structures M, if M ⊨ ϕ , then M ⊨ K_i ϕ. In other words, our agent knows all the tautologies. This of course holds for tautologies only… i.e., ϕ ⇒ Kϕ is not valid (unless we axiomatize it).

Which is the relation between knowing something and the truth value of that?

2) **Axiom A2: Knowledge Axiom:** :If an agent knows ϕ , then is true ϕ, K$\phi \Rightarrow \phi$.

This is considered to be the central property of Modal logic for knowledge. (valid only if the possibility/accessibility relation is reflexive).

What about belief? This axiom, in belief logic, does not hold. It is usually substituted by: $\neg K_i false$

What about the own knowledge of an agent?

3) **Axiom A3: Positive Introspection Axiom: $K_i\phi \rightarrow K_iK_i\,\phi$.**
4) **Axiom A4: Negative Introspection Axiom**: $\neg K_i\phi \rightarrow K_i\neg K_i\,\phi$

- **A0**: All instances of propositional tautologies are valid
- **MP** Modus Ponens: if φ is valid and $\varphi \Rightarrow \psi$ is valid, then ψ is valid
- (Distribution Axiom) **A1**: $(K_i\varphi \wedge K_i(\varphi \Rightarrow \psi)) \Rightarrow K_i\psi$
- (Knowledge Generalization Rule) **G**: for all structures M, if $M \vDash \varphi$, then $M \vDash K_i\varphi$
- (Knowledge Axiom) **A2**: If an agent knows φ , then φ is true $(K_i\varphi \Rightarrow \varphi)$
- (Positive Introspection Axiom) **A3**: $(K_i\varphi \Rightarrow K_iK_i\varphi)$
- (Negative Introspection Axiom) **A4**: $(\neg K_i\varphi \Rightarrow K_i\neg K_i\varphi)$

Shall we take all of them?
- Do we need all of them?
- If not, which one will we keep?

1) A0, A1, MP, and G: **Logic K**
2) A0–A4, MP, G: **Logic S5**
3) A0–A3, MP, and G: **Logic S4**

It up to us decide: which operators, which axioms, and mix the ingredients to achieve a logic.

Linear-time Temporal Logic (LTL)

The world observed by an agent evolves through time. The framework of modal logic has been exploited to support a qualitative representation and reasoning over the evolution of worlds along the temporal dimension

- The accessibility relation now is mapped into the temporal dimension.
- Each world is labelled with an integer: time is then discrete.

Two possible ways for the world to evolve:

1) **Linear-time**: from each world, there in only one other accessible world.
2) **Branching-time**: from each world, many worlds can be accessed

Temporal logic

Which operators? Notice that we chose to organize time along a discrete evolution of worlds.

- Something is true at the next moment in time. $\bigcirc\varphi$
- There is a ϕ that is always true in the future. $\Box\varphi$
- There is a ϕ that is true sometimes in the future $\Diamond\varphi$
- There exists a moment when ϕholds and ψ will continuously hold from now until this moment: $\phi U\ \psi$
- ϕ will continuously hold from now on unless ψ occurs, in which case ϕ will cease: $\phi W\ \psi$

Semantics

The semantics is defined again starting from a Kripke strcuture M. However:

- The accessibility relation is mapped on a linear structure, where each world/state is simply labelled with a natural number…
- … thus indicating the state or the number is the same.

A proposition is entailed when:

- $(M,i) \vDash p$ iff $i \in \pi(p)$
- $(M,i) \vDash \bigcirc\varphi$ iff $(M,i+1) \vDash \varphi$
- $(M,i) \vDash \Box\varphi$ iff $(M,j) \vDash \varphi,\ \forall j \geq i$

- $(M,i) \vDash \varphi\mathcal{U}\psi$ iff $\exists k \geq i\ s.t.\ (M,k) \vDash \psi$ and $\forall j.\ i \leq j \leq k\ (M,j) \vDash \varphi$
- $(M,i) \vDash \varphi\mathcal{W}\psi$ iff either $(M,i) \vDash \varphi\mathcal{U}\psi$ or $(M,i) \vDash \Box\varphi$

Example

- Alice is in a room, and will toss a coin, the coin will land head or tail.
- Bob is in another room, he will hear the sound of the coin on the floor, will enter the room and observe if the coin is head or tail.

M=(S, π, K_a, K_b):

- $\Phi = \{tossed, head, tail\}$
- Three stages, five worlds $S = \{s_0, h_1, t_1, h_2, t_2\}$
- $\pi(tossed) = \{h_1, t_1, h_2, t_2\}$
- $\pi(head) = \{h_1, h_2\}$
- $\pi(tail) = \{t_1, t_2\}$
- $K_A = \{(s,s) \mid s \in S\}$
- $K_B = \{(s,s) \mid s \in S\} \cup \{(h_1,t_1),(t_1,h_1)\}$

$$(M,0) \vDash \bigcirc tossed \wedge K_B \bigcirc (heads \vee tail) \wedge K_B \Box(tail \Rightarrow K_A tail)$$

Model checking

TL (but also CTL) lend themselves nicely to the modelling of system specifications that can be considered as finite state machines. They proved quite effective also in modelling distributed systems, where different agents can exchange messages and information.

A question arises: given a specification of a (distributed) system, is it possible to prove that some properties always/sometimes hold? •

- Is it possible to prove that a certain state will never occur? (safety properties)
- Is it possible to prove that a certain state will occur, sooner or later?
- Is it possible to prove that my system will not end up in a loop? (livelock/deadlock)

Model Checking is a discipline that investigated the possibility of proving such formal properties.

Knowledge and Beliefs

Semantic Networks

Historically, the first proposal of a Semantic Network is due to Peirce (1909).

Many proposals. mainly based on a graphical language. A simple one:

- objects and categories represented by their name (a label) into ovals or boxes…
- … connected by links, also labelled Particular importance is assumed by the link labels, that characterise the nature of the link.

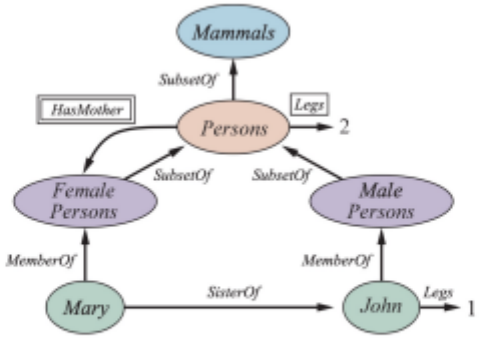


Figure 10.4 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

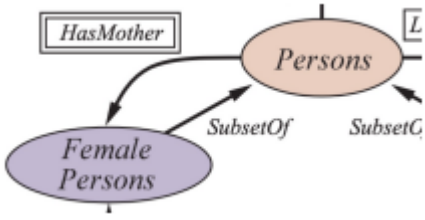
• Categories and Objects represented with the same symbol…

• Four different types of links, with the same graphical representation

Four different types of links, with the same graphical representation:

- Mary, John, and SisterOf: relation between objects
- Persons have 2 legs: is it referred to the category?
- Mammals, Persons, and SubsetOf: is-a like relation
- hasMother relation: which meaning?

-The category of Persons does not have mothers… indeed it's a category!



The intended meaning is more complex: each member of Persons has a mother who belongs to the category Female Persons. Relation between the two classes. Link different from the other ones.

$$\forall\, x, x \in Persons \Rightarrow$$
$$[\forall\, y\ HasMother(x, y) \Rightarrow y \in \text{FemalePersons}]$$

Summing up, there are:

1) properties about the categories as a whole
2) properties about the members of the categories

Semantic Networks: Single inheritance and properties

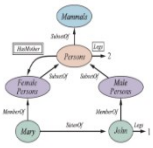


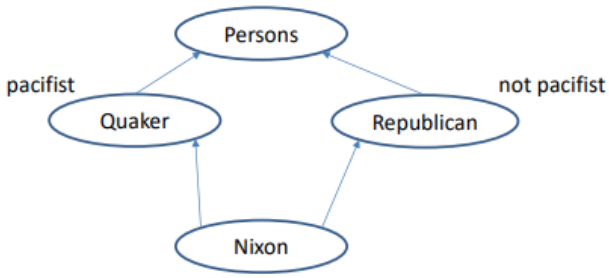
Figure 10.4 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

Reasoning about inheritance and properties is easy:

1) start from your object: does it exhibit the property?
2) if not, look at the category it belongs: does it exhibit the property? •
3) if not, move up to the next category in the hierarchy, and check it again…
4) if needed, iterate previous step.

Semantic Networks – multiple inheritance and properties

Things are nasty when allowing multiple inheritance… the "diamond" problem:



Is Nikon a pacifist? a non-pacifist? It can be both (incoherence)…

OOP Languages suffer the problem, when dealing with overriden methods:

- C++ allows for multiple inheritance, but you have to specify for an instance if you are viewing it from the perspective of one class, or another
- Java and C# do not allow multiple inheritance (but allow multiple interfaces)

Semantic Networks: Few limits

FOL is surely more powerful

- negation?
- universal and existential quantified properties
- disjunction?
- nested function symbols?

We could extend the graphical language of semantic networks… but at the cost of simplicity.

Semantic Networks: an advantage on default and overriding

Default properties can be assigned directly to the categories.

It is immediate to specify exceptions:

- per-categories.
- per-instances E.g.: Legs(2) property.

Semantic Networks: Procedural attachment

Many semantic networks systems allow to attach a call to a special procedure/algorithm, whenever special cases should be treated differently, and not by using the general inference algorithm.

Typically, the procedural attachment has exploited to invoke "pieces of programs".

- powerful, and very easy to be used.
- loss of declarative (logical) meaning.

Frames

Proposed by Minsky in 1975. Similar to Semantic Networks. Adopted in Expert Systems, together with rules.

Here is the essence of the theory: When one encounters a new situation (or makes a substantial change in one's view of the present problem) ***one selects from memory a structure called a Frame****. This is a remembered framework to be adapted to fit reality by changing details as necessary.*

A frame is a data-structure for representing a stereotyped situation*, like being in a certain kind of living room, or going to a child's birthday party.* ***Attached to each frame are several kinds of information****. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.*

Minsky, 1974.

Def: Frame: a frame is a piece of knowledge that describes an object in terms of its properties:

- it has a (unique) name.
- each property is represented through couples slot-filler.

Examples:

```
(tripLeg123                    (toronto
      <:Instance-of TripLeg>         <:Instance-of City>
      <:Destination toronto>         <:Province ontario>
      ...                            <:Population 4.5M>
)                                    ...
                               )
```

In terminological approaches (DL) an object is an instance of a category on the basis of sufficient conditions (logical approach).

In the Frames proposal, an object belong to a category if it is similar enough to some typical members of the category, named prototypes.

- Prototypes allow to reason about general properties of the (objects belonging to the) categories.
- Exceptions are allowed, for objects that exhibit different property values (Defeasible).
- Default values are used when "real" values are missing.
- Links between frames are though property values.

Frames and procedural information

Slots can contain additional information about the fillers, named facets:

- • default value.
- Type
- Allowed range.

More interestingly:

- fillers can take the form of procedural attachments.
- … that can be activated by specific facets.

Slots can contain additional information about the fillers:

- fillers can take the form of procedural attachments.
- … that can be activated by specific facets.
- looking for the value of a slot: if-needed.
- adding values: if-added.
- deleting values: if-removed

The Frames proposal can be viewed as an attempt to integrate declarative and procedural knowledge.

Introduction to Description Logic(s)

Def ambiguity : In natural language many nouns can be used to refer as category or as relations

E.g., child can be used:

- *As a category: a person of a young age*
- *To indicate a relation: the inverse of parent*

Eg: In italian,, we can use the term "professore":

- *as the category of people teaching.*
- *as the relation between people in this room and the guy is talking now.*

Again, we are really interested in the generalization relation. We would like to define some generalization relation by ourselves but we would like also to automatically infer generalization hierarchies as a consequence of the description we have made of concepts. We would like to represent complex concepts as the results of some "composition" of simpler concepts we would like to know if an individual belongs to some category or not.

Description Logic is a (family of) logic that focus on the description of the terms.

DL vs FOL

FOL focuses on sentences.

FOL does not help you on reasoning on complex categories.

Example

We can say that X is a hunter by a 1-ary predicate Hunter(X); similarly, we can say Gatherer(X). What if we want to say that X is both a hunter and a gatherer?

- *We could add a new predicate Hunter&Gatherer(X).*
- *Then, we should add an axiom to relate the three predicates: Hunter&Gatherer(X) ↔ Hunter(X) ∧ Gatherer(X).*

We should do this for every compound concept for every possible relation among them that we want to capture.

Example:

A biped is an object with exactly two legs attached to a body.

Biped(a) ⇒ ∃ l1, l2, b t.c : Leg(l1) ∧ Leg(l2) ∧ Body(b)∧ PartOf (l1, a) ∧ PartOf (l2, a) ∧ PartOf (b, a) ∧ Attached(l1, b) ∧ Attached(l2, b) ∧ l1≠ l2 ∧ [∀l3 Leg(l3) ∧ PartOf (l3, a) ⇒ (l3 = l1 ∨ l3 = l2)] (the last part is only for defining that a bipede has two legs).

Representing the "exactly two legs" information appears to be quite clumsy.

- In Frames the generalization relation is all user defined!
- Roles can have multiple fillers, while slots can't (at least in basic Frames systems)
- Frames provide a procedural solution to the creation and instantiation of individuals

A simple logic: DL

Two different sets of symbols: logical symbols (with a fixed meaning) and non-logical symbols (domain-dependent).

Def: logical symbols:they have a fixed meaning

- Punctation: (,),[,]
- Positive integers
- **Concept-formin operators** : ALL, EXISTS, FILLS, AND
- **Connectiveness**: ⊑, .=, →

Def : non logical symbols: domain dependent meaning

- **Atomic concepts**: Person, FatherOfOnlyGirls (camel casing, first letter capital, same as OOP)
- **Roles**: :Height, :Age, :FatherOf (same as concepts, but precede by columns(duepunti))
- **Constants**: john, federicoChesani (camel casing, but starting with uncapitalized letter)

Def Complex Concepts: Complex Concepts can be created by combining atomic concepts together, using the **concept forming operators**.

- Every atomic concept is a concept;
- 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.

What about sentences?

- 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.

A KB in a description logic like DL is considered to be any collection of sentences of this form.

- **Constants** stand for **individuals** in some application domain.
- **Concepts** stand for **classes or categories** of individuals.
- **Roles** stand for **binary relations** over those individuals.

In many research area there is a distinction between:

- **A-Box**, Assertion Box: a list of facts about individuals.(Federico is bold).
- **T-Box**, Terminological box: a list of sentences (also called axioms) that describes the concepts.(every man is mortal).

In the Semantic Web initiative, such distinction is not so stressed. It happens you can have the former, the latter, or more often, a combination of both.

Concept forming operators

A desired feature in a description logic is to **define complex concepts in terms of more simpler concepts**. This is achieved by means of the concept-forming operators. We just introduced:

- [ALL r d]
- [EXISTS n r]
- [FILLS r c]
- [AND d1 . . . dn]

There are many other concept-forming operators.

Def: All: Stands for those individuals that are r-related only to individuals of class d.

- **ALL :HasChild Male:** All the individuals that have zero or more children, but all males.
- **ALL :HaveStudent Male** All the individuals that have only male students (o no students at all). In this case with term individuals we could intend universities, schools, courses.
- **ALL : AreInRoom RoomWhereFedericoIsCurrently** All the individuals in this room.

Def: Exists n r : It stands for the class of individuals in the domain that are related by relation r to at least n other individuals.

- **[EXISTS 1 :Child]** All the individuals (the class of the individuals) that have at least one child
- **[EXISTS 2 :HasCar]** All the individuals that have at least two cars
- **[EXISTS 6 :HasWheels]** All the individuals that have at least six wheels (individual? in which sense? mind the term!)

Def: FILLS r c: Stands for those individuals that are related r-related to the individual identified by c.

- **[FILLS :Child francescoChesani]** All the individuals that have has child Francesco Chesani.
- **[FILLS :HasCar aa123bb]** All the individuals that have the car with plate aa123bb.
- **[FILLS :AreAttendingTheCourse thisCourse]** All the individuals that are attending this course.

Def: [AND d1 . . . dn] : : Stands for anything that is described by d1 and by . . . dn. Each individual is a member of all the categories d1 . . . dn. If we refer to the notion of sets, here we have the idea of **intersection.**

```
[ AND
    Company
    [ EXISTS 7 :Director]
    [ ALL :Manager [ AND
        Woman
        [ FILLS :Degree phD ]
    ] ]
    [ FILLS :MinSalary $24.00/hour]
]
```

Example 1:

The City Council of Bologna supports some families with cash discounts for the children schools. Such families must have at least three children, and at least one of the parents must be unemployed. Try to model such families using the operators ALL, EXISTS, FILLS, AND.

- *which atomic concepts?*

- *do we need to use some data types, besides the individuals?*
- *which complex concepts? which relations/roles?*

Atomic concepts:

- *Person?*
- *Children / Parent?*
- *Employed / Unemployed?*

Relations :

- *:MemberOf / :HasMember ?*
- *:ChildOf / :ParentOf ?*
- *:Employed / :Unemployed ?*

Family – a sad solution...

```
[ AND
  [ ALL :HasMember Person ]
  [ EXISTS 3 :HasMember ]
  [ ALL :HasMemberChild Person ]
  [ EXISTS 1 :HasMemberChild ]
  [ ALL :HasMemberParent Person ]
  [ EXISTS 2 :HasMemberParent ]
]
```

Family – better version...

```
[ AND
  [ ALL :HasMember Person ]
  [ EXISTS 3 :HasMember ]
  [ ALL :HasMemberChild Person ]
  [ EXISTS 1 :HasMemberChild ]
  [ ALL :HasMemberParent [AND
      Person
      EXISTS 1 :HasChild
      ALL :HasChild Person
  ]
```

Considerations...

- What about the concept of employed? More troubling, what about unemployed?
- How many parents can have a family?
- Why we do not define the concept of Parent, and then use it inside Family?
- Any relation between the :HasMember and the :HasMemberChild relations?
- Any relation between the parents and the children?

Is it possible that this logic is too poor? What else we would like to have?

Sentences

Sentences are expression that are intended to be true or false in the domain.

Def: d1 ⊑ d2: Concept d1 is subsumed by concept d2, i.e. every individual that satisfies d1 satisfies also d2.

Example: PhDStudent ⊑ Student: Every phd student is also a student (but not vice-versa).

Def: d1 = d2 Concept d1 is equivalent to concept d2, i.e. the individuals that satisfy d1 are precisely those that satisfy d2.

Example: PhDStudent .= [AND Student Graduated HasFunding] A phd student is a student that already graduated, and that has some funding.

Def: c → d: The individual denoted by c satisfies the description expressed by concept d.

Example: federico → Professor Federico is an instance of category Professor.

Interpretations

An Interpretation $\mathfrak{I}$ is a pair (D, I) where:

- D is any set of objects, called domain
- I is a mapping called the **interpretation mapping**, from the non-logical symbols of DL to elements and relations over D:
  1 for every constant c, I[c] ∈ D.
  2 for every atomic concept a, I[a] ⊆ D.
  3 for every role r, I[r] ⊆ D × D.

What is the interpretation of complex concepts?

- for the distinguished concept Thing, I[Thing] = D
- I[**[ALL r d]]** = {x ∈ D | for any y, if ⟨x, y⟩ ∈ I[r], then y ∈ I[d]}
- I[**[EXISTS n r]]** = {x ∈ D | there are at least n distinct y such that ⟨x, y⟩ ∈ I[r]}
- I**[[FILLS r c]]** = {x ∈ D | ⟨x, I[c]⟩ ∈ I[r]]}; I[[AND d1 . . . dn]] = I[d1] T . . . T I[dn]

Given an interpretation, when a sentence is true?

Given an interpretation $\mathfrak{I}$ = (D, I), a sentence α is true ($\mathfrak{I}$ |= α), according to the following:

1 $\mathfrak{I}$ |= (c → d) iff I[c] ∈ I[d].
2 $\mathfrak{I}$ |= (d ⊑ d ') iff I[d] ⊆ I[d '].
3 $\mathfrak{I}$ |= (d .= d ') iff I[d] = I[d ']

We use the notation $\mathfrak{I}$ |= S, where S is a set of sentences, to mean that all the sentences in S are true in $\mathfrak{I}$.

If there exists an interpretation $\mathfrak{I}$ such that $\mathfrak{I}$ |= S, we say that $\mathfrak{I}$ is a **model** of S.

Entailment

Entailment is defined as in First Order Logic:

Def: logically entail : A set S of sentences logically entails α if for every interpretation $\mathfrak{I}$, if $\mathfrak{I}$ |= S then $\mathfrak{I}$ |= α.

Reasoning on the TBox

In description logic the are some fundamental questions that we would like to get an automatic answer to. Given a knowledge base expressed as a set S of sentences:

1 **Satisfiability:** A concept d is **satisfiable** with respect to S if there exists an interpretation $\mathfrak{I}$ of S such that I[d] is nonempty. In such cases, $\mathfrak{I}$ is a **model** of d.

2 **Subsumption**: A concept d is **subsumed** by a concept d ' w.r.t to S if I[d] ⊆ I[d '] for every **model** $\Im$ of S.
3 **Equivalence**: Two concepts d and d ' are **equivalent** with respect to S if I[d] = I[d ']for every model $\Im$ of S.
4 **Disjointness**: . . .

Reduction to Subsumption

Satisfiability, Equivalence, and Disjointness can be computed by reducing them to the only concept of subsumption. The following proposition hold:

Def: Reduction to Subsumption: For concepts d and d ' :

- d is **unsatisfiable** ⇔ d is **subsumed** by ⊥.
- d and d ' are **equivalent** ⇔ d is **subsumed** by d ' and d ' is subsumed by d.
- d and d ' are **disjoint** ⇔ d ∩ d ' is subsumed by ⊥

Reasoning on the ABox

What about the individuals? We would like to answer also to the question: does a constant c satisfies concept d?

With respect to a set S of sentences, this amounts to ask: S |= (c → d)?

Also such question can be reduced to the concept of subsumption. . .

Which type of reasoning?

Summarizing, two fundamental questions that we would like to get an automatic answer to. Given a knowledge base expressed as a set S of sentences:

1 does a constant c satisfies concept d?
2 Is a concept d subsumed by a concept d ' ?

Answering to these questions amount to compute the entailment for the subsumption problem.

We would like to answer these questions independently by the specific domain or interpretation. . .

Computing Subsumption

Since all the important questions over a set of sentences S can be reduced to test the subsumption property between two concepts, such task is of the utmost importance, from both the aspects of soundness/completeness, and from computational costs.

Two main algorithms families for computing subsumption:

- based on structural matching,
- tableaux-based algorithms.

Reasoning through Structural Matching

We want to prove:

$$KB \models (d \sqsubseteq e)$$

The main idea is to:

1 Put d and e in a special normalized form.

2 Determine whether each part of the normalized e is accounted for by some part of the normalized d.

INSERISCI IMMAGINE DISEGNO ALLA LAVAGNA DI NICO

Structural Matching algorithm:

Input: Two normalised concepts d and e where d is [**AND** $d_1 \dots d_m$] and e is [**AND** $e_1 \dots e_{m'}$]
Output: Yes or No, according to $KB \models (d \sqsubseteq e)$
Return Yes iff for each component e_j, for $i \leq j \leq m'$, there exists a component d_i, $1 \leq i \leq m$ such that *di matches* e_j as follows:

- if e_j is an atomic concept, then d_i must be identical to e_j;
- if e_j is of the form [**FILLS** r c], then d_i must be identical;
- if e_j is of the form [**EXISTS** n r], then d_i must be [**EXISTS** n' r], for some $n' \geq n$; if $n = 1$, then d_i can be also of the form [**FILLS** r c] for any constant c;
- if e_j is of the form [**ALL** r e'], then the corresponding d_i must be of the form [**ALL** r d'], and $d' \sqsubseteq e'$.

Reasoning through Tableaux-based algorithms

We want to prove:

KB |= (d ⊑ e)

To do so, we will exploit the following result:

KB |= (C ⊑ D) ⇔(KB ∪ (x : C ⊓ ¬D)) is not consistent (with x new concept name).

Used by many reasoners: FaCT, FaCT++ Racer Pellet.

Intuition:

IMMAGINE DDI NICOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

Construct a representation of (KB ∪ (x : C ⊓ ¬D)), by “explicating additional constraints on the model that are implied by the concepts in ABox and the axioms in TBox”, until A, ¬A (inconsistency) is achieved.

Examples of *expansion rules*:

⊓-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

⊔-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

∃-rule: if 1. $\exists r.C \in \mathcal{L}(x)$, x is not blocked, and
2. x has no r-successor y with $C \in \mathcal{L}(y)$,
then create a new node y with $\mathcal{L}((x, y)) = \{r\}$ and $\mathcal{L}(y) = \{C\}$

∀-rule: if 1. $\forall r.C \in \mathcal{L}(x)$, x is not blocked, and
2. there is an r-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

⊑-rule: if 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x is not blocked, and
2. $C_2 \sqcup \neg C_1 \notin \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \neg C_1\}$

Figure 3.1: The tableau expansion rules for $\mathcal{ALC}$.

Closed- vs. open-world semantics

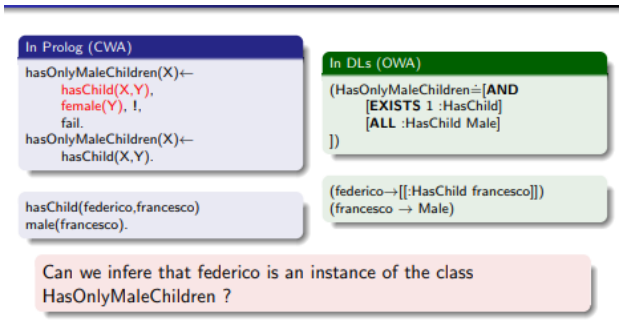
Differently by many other formalisms, Description Logics are based on an Open World Assumption.

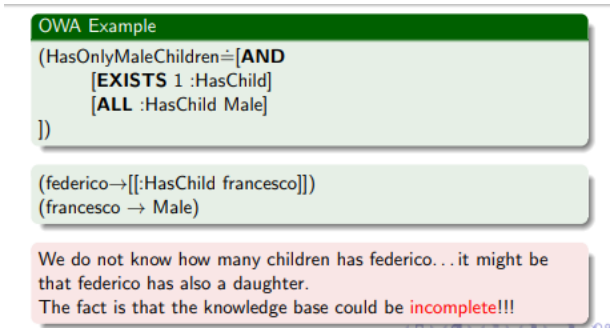
<u>If a sentence cannot be inferred, then its truthness value is unknown.</u>

Note: somehow this characteristics is linked to the idea of distributed information in the Web.

Sons of Gianfranco are Federico and Paolo. But the council of Bologna doesn't says if Gianfranco had other children in the world. I was thinking like a close world assumption but is an open world.

Example:

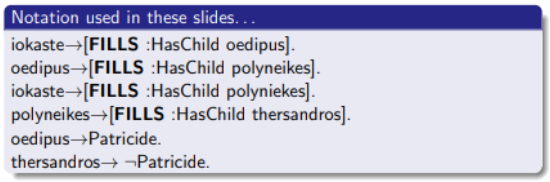




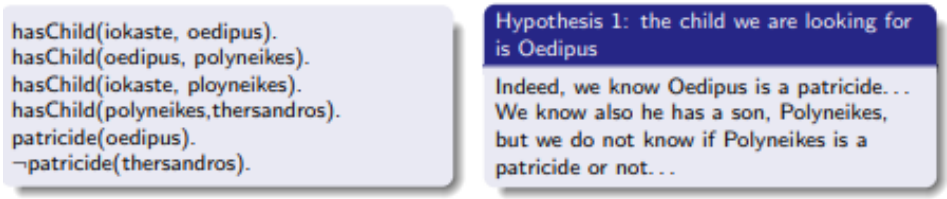
The "Oedipus example":

Oedipus killed his father, married his mother Iokaste, and had children with her, among them Polyneikes. Finally, also Polyneikes had children, among them Thersandros.

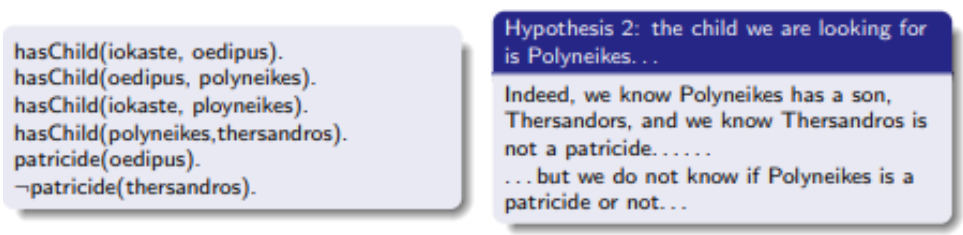
<u>We want to know if Iokaste has a child that is a patricide and that itself has a child that is not a patricide. .</u> .



Hypothesis 1:



Hypothesis 2:

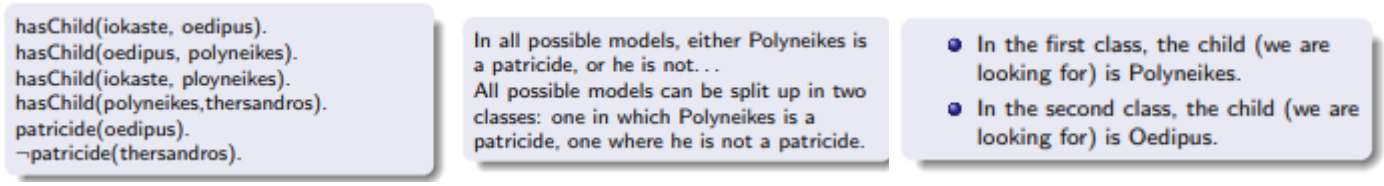


Orginal question: if Iokaste has a child that is a patricide and that itself has a child that is not a patricide.

We could infer that under the OWA, the answer is "the KB does not entail the sentence."

BUT THIS REASONING IS NOT CORRECT!

Possible solution:



*Thus: In ALL models Iokaste has a child that is a patricide and that itself has a child that is not a patricide. The answer to the previous question is **YES**.*

In open world assumption you have to take in mind this example. Contro intuitive. Whenever you don't know something you have for find in the space of possibilities if what you are considering is possible or not.

Extending DL

It is possible to extend the presented description logic in several directions:

- by adding concept-forming operators
- by relating roles, and considering also complex roles
- by adding **rules**.

Bounds on the number of roles fillers

We have already the operator **EXISTS**. We could similarly add the operator **AT-MOST**, where [**AT-MOST** n r] describes individuals related by role r to at most n individuals.

Example:

[AT-MOST 1 :Child] denotes all the parents that have only one child.

This extension could be dangerous: What is the meaning of the following concept: [AND [EXISTS 4 r] [AT-MOST 3 r]] How do we treat such situation?

Sets of individuals

It would be nice to specify that a role can be filled only by individuals belonging to a certain set (without recurring to a concept). We could add the operator **ONE-OF**, where [**ONE-OF** c1c2 . . . cn] is a concept satisifed only by ci , used in conjunction with **ALL** would lead to a restriction on the individuals that could fill a certain role.

Example

(Beatles .=[ALL :BandMember [ONE-OF john paul george ringo]])

Implicitly this means that. . . . . there is an AT-MOST restriction limited to 4 on the role :BandMember

.

Qualified Number Restrictions

What if we want to describe, e.g., those parents that have at least 2 male children? We can add the operator [**EXISTS** n r d], meaning all the individuals that are r-related to at least n individuals that are instances of concept d.

*Example: [**EXISTS** 2 :Child Male]*

Unfortunately, this simple extension increase the computational complexity of entailment (subsumption) . .

Other Logic operators

We have completely forgot standard usual operators such as:

- **OR**: what if we want to describe all the young and the elder people, but not the adults?
- ¬: what if we want to describe all the people that is not instance of a concept d?

What happens to the computational costs? What happen to the decidability?

Relating the Roles

What if we want to relate the fillers of certain roles? Suppose you want to say that in a small company the CEO and the owner must be the same individual. . .

We can add the operator **[SAME-AS** r1 r2], which equates fillers of roles r1 and r2.

Example [**SAME-AS** :CEO :Owner]

Unfortunately, also this simple extension increase the computational complexity of entailment, and if allowed with role chains, can lead to undecidability . . .

Complex Roles

Until now we treated roles ad primitive concepts. . . what if, for example, we want to consider a role defined as the conjunction of more roles?

And, more interesting, what if we want to add ideas like the **inverse** of a role? *Saying for example that :Parent is the inverse of :Child? In particular, this type of information is heavily used within the Semantic Web initiative. . .*

Rules

In the language presented here, there is no way of saying, for example, that all instances of one concept are also instances of another.

- We could this by adding definitions of the type (d1 .= d2), where d1 and d2 are complex concepts with their own definition.
- But this could lead to several classification problems, when computing subsumption.

Some description logics allows the introduction of rules, like for example:

(**if** d1 **then** [**FILLS** r c]).

Meaning that every individual of class d1 also has the specified property.

How many logics?

Summing up:

- we started presenting a language, we called it DL.
- we provided its semantics
- we have seen it is nice, but that there could be other nice constructs that could help us. . .

Where are we going exactly?

An entire family of logics

Description logics is a family of logics. Eahc logic is different depending on which **operators** are admitted in the logic. Of course, more operators means:

- higher expressivity.
- higher computational costs
- the logic it might result to be undecidable.

At http://www.cs.man.ac.uk/~ezolin/dl/ there is a nice application that shows complexity issues and decidability depending on which operator you decide to include/exclude in your logic.

A description logic is named upon the operators included.

A minimal, yet useful logic is named AL: Attributive Language. It includes:

- Atomic concept.
- Universal concept (Thing or $\top$)
- Bottom concept (Nothing or $\bot$)
- Atomic negation ($\neg$A), applied only to atomic concepts
- **AND** operator (also called intersection $\sqcap$)
- **ALL** operator (also called value restriction $\forall$R.C)
- [**EXISTS** 1 r] operator (also called limited existential quantification $\exists$R.C)

A description logic is named upon the operators included. . .

- ALC extends AL with the negation for concepts (C stand for Complement).
- S is synonim of ALC augmented with transitive roles.

Following this line, a set of letters indicate the expressivity of the logic and name the logic

- F Functional Properties
- E Full existential Qualification
- U Concept Union
- C Complex Concept Negation
- S is synonim of ALC augmented with transitive roles
- H Role Hierarchy.
- R Limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness
- O Nominals
- I Inverse Properties
- N Cardinality Restrictions
- Q Qualified Cardinality Restrictions
- (D) Use of datatype properties, data values or data types.

When choosing a knowledge representation formalism, there is always a trade-off between performances and expressivity. The W3C working group defined the Ontology Web Language (OWL), with a set of operators for representing the knowledge. OWL comes out in three different flavour:

- OWL-Lite, is based on SHIN^(D).
- OWL-DL, based on SHOIN^(D).
- OWL-Full, highly expressive, can be also high-order logic.

Constructor	DL syntax	Example
intersectionOf	$C_1 \sqcap \cdots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \cdots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1 \ldots x_n\}$	{john, mary}
allValuesFrom	$\forall P.C$	∀hasChild.Doctor
someValuesFrom	$\exists r.C$	∃hasChild.Lawyer
hasValue	$\exists r.\{x\}$	∃citizenOf.{USA}
minCardinality	$(\geqslant nr)$	(⩾ 2 hasChild)
maxCardinality	$(\leqslant nr)$	(⩽ 1 hasChild)
inverseOf	r^-	hasChild$^-$

OWL-DL Constructors

- **intersectionOf**: a concept is defined as conjunction (AND) of other concepts. E.g.: Human ⊓ Male.
- **unionOf**: a concept is defined as the disjunction (OR) of other concepts. E.g.: Tall ⊔ Nice.
- **complementOf**: a concept is defined as all the individuals that are not members of a given concept. E.g.: ¬Male stands for all the females. . .
- **oneOf**: a concept is defined as a specified set of individuals. E.g.: TeachersOfThisCourse .= { federico, paola, marco}.
- **allValuesFrom**: all the individuals that are in relation r only with individuals of a certain class. E.g.: ∀hasChild.Male
- **someValuesFrom**: all the individuals that are in relation r with at least one individual of a certain class. E.g.: ∃hasChild.Male
- **hasValue**: all the individuals that are in relation r only with a certain individual. E.g.: ∃hasChild.{francesco}
- **minCardinality**: all the individuals that are in relation r with at least n individuals. E.g.: all the parents that have at least 4 children (≥ 2 hasChild)
- **maxCardinality**: all the individuals that are in relation r with at most n individuals. E.g.: all the parents that have at most 4 children (≤ 2 hasChild)
- **inverseOf**: relations have a direction from a domain to a range. It is common to define also the property from the range to the domain. It is useful to define that a relation is the inverse of another one. E.g.: hasChild inverseOf hasParent. If we have (federico hasChild francesco), then it also holds (francesco hasParent federico)

OWL-DL Axioms

FunctionalProperty: if a relation (property) r is defined functional, there can be only one value y for each x in relation r. I.e., there cannot be two distinct y1 and y2 such that we have (x r y1) and (x r y2). E.g., consider the relation isToppingOf, and consider the Parma ham slice s1, and two pizzas p1 and p2. If we have (s1 isToppingOf p1) and (s1 isToppingOf p2), then we can conclude two different things: p1 and p2 are the same pizza. . . if I stated previously that p1≠ p2, then my KB is inconsistent.

Which tools for the Web?

After OWL has been defined, several tools have been developed, in particular to support OWL-DL.

E.g:

- *Prot´eg´e ontology editor,supports SHOIN^(D)*
- *Pellet, Racer and FaCT++ are reasoners that supports SHOIN^(D)*

OWL-DL is not the only choice. For example, the ontology SnoMed is based on EL with additional role properties.

Ontologies and Protegé

An ontology is a formal, explicit description of a domain of interest

Allows to specify:

- Classes (domain concepts)
- Semantic relation between classes/properties associated to a concept (slots, and possibly restrictions, facets)
- Possibly, a further logic level (axioms and rules)

Classes instances Ontology + Istances = Knowledge Base

Starting point: a deep analysis, complete(?) and correct(?) of an application domain. The knowledge can be then represented/formalized by means of an ontology. Uses:

- Export
  - Application export
  - Knowledge export
- Modelling
  - Fundamental in every step of teh developing process
- Interoperability between different applications.

Ontologis and Semantic Web

Two proposals within the SW Initiative:

- RDF Schema (RDFS), RDF extensions with proper terms for ontological concepts
- OWL (Ontology Web Language), extend RDFS
  - OWL Lite
  - OWL DL
  - OWL Full

Based on Description Logics

OWL and Description Logic

- Description Logics are a family of logics
- Each fragment depend on which operators are supported in the logic
- More supported operators-> higher the complexity
- OWL-DL supports the following operators:

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

Developing an ontology

A possible development process:

1. Analyse the **domain** and the **goal** of the ontology
2. Consider to **reuse** existing ontologies
3. Determine the key **concepts** of the domain
4. Organize concepts in classes and **hierarchies** among classes
5. Determine the **properties** of the classes
6. Add **constraints** (allowed values) on the properties
7. Create the **instances**
8. Assigns **values** to the properties for each instance

Protégé: Developing an ontology

Editing function are split in different tabs:

- Tab “Classes”: classes editor
- Tab “Object Properties”
- Tab “Data Properties”
- Tab “Individuals”
- Tab “Forms”: Allows to define forms for the data insertion phase

Protégé: Classes editor

Assertions:

- **Necessary and Sufficient** (complete definition) (C1 equivalent to C2)
- **Necessary** (partial definition) (C1 has superclass C2)
- Inherited
- Assertions are about **Properties** and **Restriction on properties**
- Disjoints (individuals of this class cannot be also individuals of another class)
    - By default, OWL classes have some “overlap”!!!!!!

Protégé: Properties

Properties are **binary relations** between two **things**.

- **Object properties**: relation between two individuals
- **Datatype properties**: relation between an individual and a primitive data type
- **Annotation properties** (metadata…)

Protégé: Inverse properties

For each object property it is possible to specify the inverse **property**.

E.g.:

- individuals: federico, Francesco
- Properties: hasParent, hasChild
- Sentence: “francesco hasParent federico”

Now, define hasChild as inverse property of hasParent:

- It is possible to automatically infer that: “federico hasChild francesco”

Functional Properties

Functional Properties: a property is functional if, for a given individual, it can be in relation (through such property) with only another individual.

E.g.: “francesco hasFather federico”.

- Note: the fact that a property is functional is not used as a constraint, but as axiom for the inference.
- E.g., if we add “francesco hasFather chicco”, we can say:
  - federico and chicco are the same individual
  - (provided federico != chicco) the two sentences are inconsistent
- **Inverse Functional Properties**: the inverse property is functional

Transitive and Symmetric Properties

- Transitive Properties
  E.g., has Ancestor
- Symmetric Properties
  E.g., hasBrother “federico hasBrother paolo” allows to infer also “paolo hasBrother federico”
- Antisymmetric Properties (if “a rel b”, it can never be “b rel a”)
  E.g., hasChild: “federico hasChild francesco”

Reflexive Properties

- Reflexive properties: if rel is reflexive, then it always holds “a rel a”, for every indidual a.
  E.g., knows: “federico knows federico”
- Irreflexive properties (it can never be “a rel a”)
  E.g., fatherOf

Properties: Domain and Range

Properties relate individuals from a **domain** to individuals from a **range**.

- **Domain** and **range are not constraints** but, rather, **axioms** used in the inference process
  E.g.: given the classes Pizza and PizzaToppings, the relation hasTopping is defined:
  - Pizza as domain
  - PizzaToppings as range

If we add "iceCream hasTopping pepperoni", then the inference engine derives that iceCream is an instance of Pizza.

- Such behaviour can generate problems (initially…)

Note: for inverse properties, there is the "inversion" of domain and range.

Describing and Defining a class

Description of a class: **necessary** conditions for an individual to belong to a class.

Definition of a class: necessary and sufficient conditions for an individual to belong to a class.

They are descripted/defined by means of:

Expressions (conjunction/disjunction of named/anonymous classes and subClassOf relation): and (intersection), or (union) and not (complement).

Property Restrictions (in Protégé everything has been reconducted to a restriction).

Property Restrictions

Quantifier Restrictions

- Existential Restrictions: individuals that are related through property prop with **at least** an individual member of the specified class (keyword "**some**", "**someValueFrom**").
- **Universal Restrictions**: individuals that are related through property prop **only** with individuals members of the specified class (keyword "**only**", "**allValuesFrom**").

Cardinality Restrictions (**possibly qualified)**

- **min n:** individuals that are related through property prop with at **least** n other individuals
- exactly n
- max n

hasValue Restrictions: individuals that are related through property prop with a certain defined individual

Protégé – Reasoning

Computation of the "**inferred ontology**"

User defined/described ontologies are called "asserted hierarchy". Protégé can invoke a reasoner (a plug-in) to compute the "inferred hierarchy".

Consistency checking: for each class, can exists at least an individual that could belong to such class?

Warning: OWL uses the Open World Assumption § If needed, Closure Axioms

Protégé - Individuals

It is possible to define also single individuals.

DL Reasoners supports also the **classification** task

Exercise: