VU Einführung in Artificial Intelligence

SS 2024

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

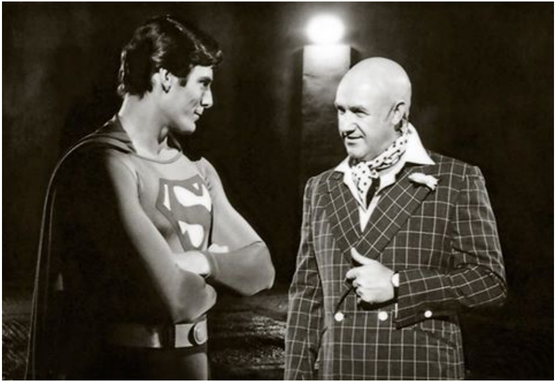
www.kr.tuwien.ac.at

Reasoning

Entailment vs. inference

- Recall that entailment is based on models while inference is based on derivations,
  - that is, entailment expresses a *semantical relation*, while
  - inference is a relation between *syntactical elements*.
- Inference is given in terms of *inference rules* in a proof system which describe the syntactic manipulation of formulas.
  - A derivation is a sequence of conclusions, sanctioned by applications of inference rules, leading to a desired goal.
  - ☞ *Theorem proving* is the general term referring to construct proofs of a desired sentence without consulting models.
- Soundness and complete relates the semantical entailment with the syntactical inference, establishing that they coincide.

Slogan



"Deductive reasoning, that's the name of the game."

–Lex Luthor (from "Superman: The Movie", 1978)

Inference rules

- A well-known inference rule, prominently used in Hilbert-type systems, is *modus ponens*:

$$\frac{\alpha \Rightarrow \beta \qquad \alpha}{\beta}.$$

  - This notation means that, whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given, then the sentence β can be inferred.

- Another useful inference rule is *$\wedge$-elimination*, as featured, e.g., in tableau systems and natural-deduction systems, which states that, from a conjunction, any of its conjuncts can be inferred:

$$\frac{\alpha \wedge \beta}{\alpha} \quad \text{and} \quad \frac{\alpha \wedge \beta}{\beta}.$$

- Clearly, above rules are sound, i.e., turn valid premisses into valid conclusions.

Inference rules (ctd.)

- In fact, each of the logical equivalences listed earlier (like commutativity and associativity of $\wedge$ and $\vee$, double-negation elimination, contraposition, etc.) can be turned into a corresponding inference rule.
  - These can then be used in derivations, generating sound inferences without the need for enumerating models.
- For example, we can use such rules for deriving $\neg\alpha$ from $(\alpha \vee \beta) \Rightarrow \gamma$ and $\neg\gamma$:
  1. Apply contraposition to $(\alpha \vee \beta) \Rightarrow \gamma$:

     F_1: $\neg\gamma \Rightarrow \neg(\alpha \vee \beta)$.
  2. Apply modus ponens to F_1 and $\neg\gamma$:

     F_2: $\neg(\alpha \vee \beta)$.
  3. Apply De Morgan to F_2:

     F_3: $\neg\alpha \wedge \neg\beta$.
  4. Apply $\wedge$-elimination to F_3:

     F_3: $\neg\alpha$.

Some properties of classical inference

The following properties hold for the inference relation $\vdash$ of classical logic, for any sound and complete proof system on which $\vdash$ is based.

➤ *Monotonicity property*:

- if $KB \vdash \alpha$ and $KB \subseteq KB'$, then $KB' \vdash \alpha$.

That is, once a conclusion α is inferred, it can never be invalidated by additional knowledge.

☞ Monotonicity means also that inference rules can be applied whenever suitable premisses are found in the knowledge base

➥ the conclusion of the rule must follow regardless of what else is in the knowledge base.

➤ *Cut rule*:

- $KB \vdash \alpha$ and $KB, \alpha \vdash \beta$, then $KB \vdash \beta$.
  (N.B. "KB, α" is a shorthand for "$KB \cup \{\alpha\}$", and likewise for more formulas instead of just α.)

Here, the proposition α serves as a "lemma" for β given KB.

Some properties of classical inference (ctd.)

- *Deduction theorem*, or *$\Rightarrow$-introduction*:
  - if $T, \alpha \vdash \beta$, then $T \vdash \alpha \Rightarrow \beta$.
- *$\Rightarrow$-elimination* (reflecting modus ponens):
  - if $KB \vdash \alpha$ and $KB \vdash \alpha \Rightarrow \beta$, then $KB \vdash \beta$.
- *$\wedge$-introduction*:
  - if $KB \vdash \alpha$ and $KB \vdash \beta$, then $KB \vdash \alpha \wedge \beta$.
- *$\wedge$-elimination*:
  - if $KB \vdash \alpha \wedge \beta$, then $KB \vdash \alpha$.
  - if $KB \vdash \alpha \wedge \beta$, then $KB \vdash \beta$.

Some properties of classical inference (ctd.)

- $\vee$ *-introduction*:
  - if *KB* $\vdash \alpha$, then *KB* $\vdash \alpha \vee \beta$;
  - if *KB* $\vdash \beta$, then *KB* $\vdash \alpha \vee \beta$.
- *Proof by cases* (or $\vee$ *-elimination*):
  - if *KB*$, \alpha \vdash \gamma$ and *KB*$, \beta \vdash \gamma$, then *KB*$, (\alpha \vee \beta) \vdash \gamma$.
- *Proof by contradiction* (*"reductio ad absurdum"*, or $\neg$*-introduction*):
  - if *KB*$, \alpha \vdash \beta$ and *KB*$, \alpha \vdash \neg\beta$, then *KB* $\vdash \neg\alpha$.
- $\neg\neg$*-elimination*:
  - if *KB* $\vdash \neg\neg\alpha$, then *KB* $\vdash \alpha$.
- *Weak* $\neg$*-elimination* (or *"ex falso sequitur quodlibet"*):
  - if *KB* $\vdash \alpha$ and *KB* $\vdash \neg\alpha$, then *KB* $\vdash \beta$.

Some properties of classical inference (ctd.)

- *$\Leftrightarrow$-introduction*:
  - if $KB \vdash \alpha \Rightarrow \beta$ and $KB \vdash \beta \Rightarrow \alpha$, then $KB \vdash \alpha \Leftrightarrow \beta$.
- *$\Leftrightarrow$-elimination*:
  - if $KB \vdash \alpha \Leftrightarrow \beta$, then $KB \vdash \alpha \Rightarrow \beta$;
  - if $KB \vdash \alpha \Leftrightarrow \beta$, then $KB \vdash \beta \Rightarrow \alpha$.
- *$\forall$-introduction*:
  - if $KB \vdash \alpha(x)$, then $KB \vdash \forall x\, \alpha(x)$, providing KB has no free occurrence of x.
- *$\forall$-elimination*:
  - if $KB \vdash \forall x\, \alpha(x)$, then $KB \vdash \alpha(t)$, where t is a term s.t. no variable of it becomes bound in $\alpha(t)$, and $\alpha(t)$ results from $\alpha(x)$ by substituting t for x.

Some properties of classical inference (ctd.)

- $\exists$-*introduction*:
  - if $KB \vdash \alpha(t)$, then $KB \vdash \exists x\, A(x)$,under the same circumstances for t and $\alpha(t)$ as in $\forall$-elimination.
- $\exists$-*elimination*:
  - if $KB, \alpha(c) \vdash \beta$, then $KB, \exists x\, \alpha(x) \vdash \beta$, where c is a constant not occurring in KB, $\alpha(x)$, and γ.

Resolution

An important proof method is *resolution*, first introduced by John Alan Robinson in 1965 with the aim for mechanisation on a computer.

- Consequently, the syntax and inference rules were kept minimal.
- Resolution works on formulas in *conjunctive normal form* (CNF):
  - a CNF is a conjunction of *clauses*, where
    - a clause is a disjunctions of *literals*, and
    - a literal is an atomic formula or the negation of an atomic formula.
  - Two literals are *complementary* if one is the negation of the other.
  - E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$ is a CNF and B and $\neg B$ are complementary literals.

Resolution (ctd.)

- *Resolution* inference rule (for CNF):

$$\frac{\ell_1 \vee \cdots \vee \ell_i \vee \cdots \vee \ell_k \qquad m_1 \vee \cdots \vee m_j \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

  where ℓ_i and m_j are complementary literals.

  - The clause in the conclusion is called *resolvent*.

- *Unit resolution* is the special case of resolution where $n = 1$, i.e., ,

$$\frac{\ell_1 \vee \cdots \vee \ell_i \vee \cdots \vee \ell_k \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

  where ℓ_i and m are complementary.

- Example:

$$\frac{P \vee Q \qquad \neg Q}{P}$$

Resolution (ctd.)

- Resolution is sound and complete for propositional logic.
- Resolution detects *unsatisfiability* of a set of clauses:
  - A set *KB* of clauses is unsatisfiable iff there is a resolution proof of the empty clause $\Box$ from *KB*.
  - Consequently, to show that $KB \vdash \alpha$ with resolution, one negates α and tests $KB \cup \{\neg\alpha\}$ for unsatisfiability.
    - This involves putting all formulas in *KB* and the formula $\neg\alpha$ into CNF.
- ☞ For the FOL case, resolution involves *unification* and the process of putting formulas into CNF requires *Skolemisation*, i.e., where existential quantifiers are replaced by terms not appearing in the original formula.
  - E.g., a skolemised version of $\forall x \exists y P(x, y)$ is $\forall x P(x, f(x))$,
    - where $f(x)$ is the newly introduced term, called *Skolem function*.

Conversion to CNF

We illustrate to conversion process by means of the formula $B \Leftrightarrow (P \lor Q)$.

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$:
$$(B \Rightarrow (P \lor Q)) \land ((P \lor Q) \Rightarrow B).$$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$:
$$(\neg B \lor P \lor Q) \land (\neg(P \lor Q) \lor B).$$
3. Move $\neg$ inwards using De Morgan's rules and double-negation elimination:
$$(\neg B \lor P \lor Q) \land ((\neg P \land \neg Q) \lor B).$$
4. Apply distributivity law ($\lor$ over $\land$) and flatten:
$$(\neg B \lor P \lor Q) \land (\neg P \lor B) \land (\neg Q \lor B).$$

Resolution example

- We show $B \Leftrightarrow (P \vee Q), \neg B \vdash \neg P$ by resolution.
- This means we test $\{B \Leftrightarrow (P \vee Q), \neg B\} \cup \{P\}$ for unsatisfiability.
- We first transform $\{B \Leftrightarrow (P \vee Q), \neg B\} \cup \{P\}$ into a set of clauses:
$$\{(\neg B \vee P \vee Q), (\neg P \vee B), (\neg Q \vee B), \neg B, P\}.$$
- The resolution proof proceeds simply as follows:
  - From $\neg P \vee B$ and $\neg B$ we derive
$$\neg P,$$
  - and from $\neg P$ and P we then already derive the empty clause $\Box$.

Knowledge Representation

Ontological engineering

We now turn to question *how* to represent facts about the world.

- *Ontological engineering* $\Longrightarrow$ create representations of general concepts like *actions*, *time*, *physical objects*, and *beliefs*.
- Initial disclaimer: we cannot represent *everything* in the world!
  - But we will leave placeholders where new knowledge for any domain can fit in.
  - E.g., we can define the concept of an physical object, but the details of different types of objects—like robots, televisions, books, turntables, etc.—can be filled in later.

Ontological engineering (ctd.)

Further caveat: certain aspects are hard to capture with FOL.

- In particular, the principal difficulty is that almost all generalisations have *exceptions*, or hold only to a *degree*.
  - E.g.: "tomatoes are red" is a useful rule, but some tomatoes are green, yellow, or orange.

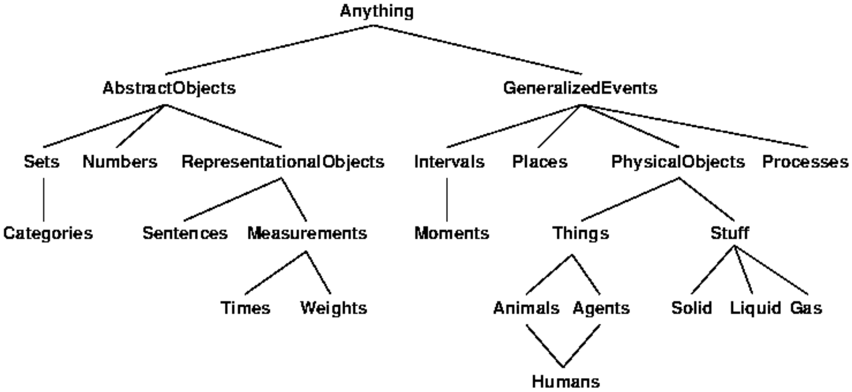
➥ Other formalisms than FOL have been designed to adequately handle such patterns:
  - *nonmonotonic logics* (like *default logic* or *circumscription*)
  - *probabilistic reasoning*

Upper ontology

- The general framework of concepts is called an *upper ontology*.

- ☞ The name derives from the convention of drawing graphs with the general concepts at the top and the more specific concepts below.

Example ontology



Categories and objects

- The organisation of objects into *categories* is a vital part of knowledge representation.

- Categories serve to make predictions about objects once they are classified.
  - One infers the presence of certain objects from perceptual input,
  - infers category membership from the perceived properties of the objects,
  - and then uses category information to make predictions about such objects.

  E.g., given an object with green, mottled skin, large size, and ovoid shape, one can infer that it is an watermelon. From this, one can further infer that it is useful for fruit salad.

Categories and objects (ctd.)

Two choices for representing categories in FOL: *predicates* and *objects*.

- E.g., for representing basketballs, we can use the predicate $Basketball(b)$, or we can *reify* the category as an object, $Basketballs$.
  - ☞ *reification:* from the Latin res, meaning thing, object
    $\Longrightarrow$ reification = "thing-ification"
- With reification, we can state $Member(b, Basketballs)$ (abbreviated as $b \in Basketballs$), i.e., b is a member of basketballs.
- We write $Subset(Basketballs, Balls)$ (abbreviated as $Basketballs \subset Balls$) to state that basketballs are a subcategory (or subset) of balls.

Inheritance and Taxonomy

- Categories serve to organise and simplify the knowledge base through *inheritance*.
  - E.g., if we say that all instances of the category *Man* are mortal, and if we assert that *Greeks* is a subclass of *Man*, then we know that all Greeks are mortal.
    - Individual Greeks (like Aristotle) *inherit* the property of mortality.

- Subclass relations organise categories into a *taxonomy*, or a *taxonomic hierarchy*.
  - Taxonomies have a centuries-long tradition in technical fields.
  - E.g., systematic biology aims to provide a taxonomy of all living and extinct species; library science has developed a taxonomy of all fields of knowledge; etc.

FOL and categories

FOL makes it easy to state facts about categories:

- An object is a member of a category:
  $BB_g \in Basketballs$
- A category is a subclass of another category:
  $Basketballs \subset Balls$
- All members of a category have some property:
  $x \in Basketballs \Rightarrow Round(x)$
- Members of a category can be recognised by certain properties:
  $(Orange(x) \wedge Round(x) \wedge Diameter(x) = 9.5'' \wedge x \in Balls) \Rightarrow x \in Basketballs$
- A category as a whole has some properties:
  $Dogs \in DomesticatedSpecies$

Other relations between categories

Two or more categories are *disjoint* iff they have no members in common:
$Disjoint(s) \Leftrightarrow \forall\, c_1, c_2\ \big((c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2) \Rightarrow$
$\quad Intersection(c_1, c_2) = \{\}\big)$

If members of a given category s constitute all elements of another category c, we have an *exhaustive decomposition*:
$ExhaustiveDecomposition(s, c) \Leftrightarrow \forall\, i\ \big(i \in c \Leftrightarrow \exists\, c_2\ \ c_2 \in s \wedge i \in c_2\big)$

A disjoint exhaustive decomposition is a *partition*:
$Partition(s, c) \Leftrightarrow (Disjoint(s) \wedge ExhaustiveDecomposition(s, c))$

Physical composition

- Objects can be part of other objects.
  - This can be expressed using the *PartOf* predicate.
- For instance:
  $PartOf(Gramatneusiedl, Austria)$;
  $PartOf(Austria, CentralEurope)$;
  $PartOf(CentralEurope, Europe)$;
  $PartOf(Europe, Earth)$.
- The *PartOf* predicate is transitive and reflexive:
  $PartOf(x, y) \land PartOf(y, z) \Rightarrow PartOf(x, z)$;
  $PartOf(x, x)$.
  - ➥ We can conclude $PartOf(Gramatneusiedl, Earth)$.

Physical composition (ctd.)

- It is also useful to define *composite objects* with definite parts but no particular structure.
- Example: we might want to say "The apples in this bag weigh one kilogram"
  - might be tempted to ascribe the weight to the set of apples in this bag, but this would be a mistake because sets have no weight.
    $\Longrightarrow$ Introduce new concept: a *bunch*.
  - E.g., if the apples are $Apple_1$, $Apple_2$, $Apple_3$, then

$$BunchOf(\{Apple_1, Apple_2, Apple_3\})$$

    denotes the composite object with the three apples as parts (not elements)

  ➥ can use the bunch as a normal, unstructured object.

Physical composition (ctd.)

We can define *BunchOf* by *logical minimisation* in terms of *PartOf*:

1. each element of s is part of $BunchOf(s)$:

   $$\forall x \;\; x \in s \Rightarrow PartOf(x, BunchOf(s)).$$

2. $BunchOf(s)$ is the smallest object satisfying Condition 1:

   $$\forall y \;\; [\forall x \;\; x \in s \Rightarrow PartOf(x, y)] \Rightarrow PartOf(BunchOf(s), y)$$

   (i.e., $BunchOf(s)$ must be part of any object that has all the elements of s as parts).

Substances and objects

➤ A significant portion of reality seems to defy *individuation*—the division into distinct objects.

- Example:
  - Suppose I have some butter and some aardvark in front of me.
  - We can say there is one aardvark but there is no obvious number of "butter-objects", as any part of a butter-object is also a butter object.

➥ We call elements defying individuation *stuff*.

☞ The distinction *things vs. stuff* corresponds to the following distinction from linguistics:

- *count nouns:* aardvarks, cars, rockets, theorems, ...
- *mass nouns:* butter, water, energy, ...

Substances and objects (ctd.)

To represent stuff in our ontology, we need to have as objects at least the gross "lumps" of stuff we interact with.

- E.g., we may recognise a lump of butter as the same butter that was left on the table yesterday.
  - In this sense, it is an object like the aardvark, say we call it $Butter_3$.

- We also define the category *Butter*—its elements are all those things of which we might say "It's butter", including $Butter_3$.
  - Any part of a butter-object is also a butter-object:
    $$(x \in Butter \wedge PartOf(y, x)) \Rightarrow y \in Butter.$$
  - We can say that butter is yellow, melts at around 30 degrees, is less dense than water, has high fat content.
  - But butter has no particular shape, size, or weight.

Substances and objects (ctd.)

Important distinction:

- *intrinsic properties:* belong to the very substance of the object, rather than to the object as a whole.
  - When you cut a substance in half, the pieces retain the same set of intrinsic properties—things like density, boiling point, flavor, color, ownership, etc.
- *extrinsic properties:* those which are *not* retained under subdivision.
  - Examples: weight, length, shape, function, etc.

Substances and objects (ctd.)

➤ We can therefore say:

- a substance, or a mass noun, is a class of objects that includes in its definition only *intrinsic* properties
- a count noun is a class that includes *some* extrinsic property in its definition.

➤ Consequently:

- The category *Stuff* is the most general substance category, specifying no intrinsic properties.
- The category *Thing* is the most general discrete object category, specifying no extrinsic properties.

☞ All physical objects belong to both categories, so the categories are *coextensive*—they refer to the same entities.