Semantic Data

Chapter 2 : Introduction to first order logic

Jean-Louis Binot

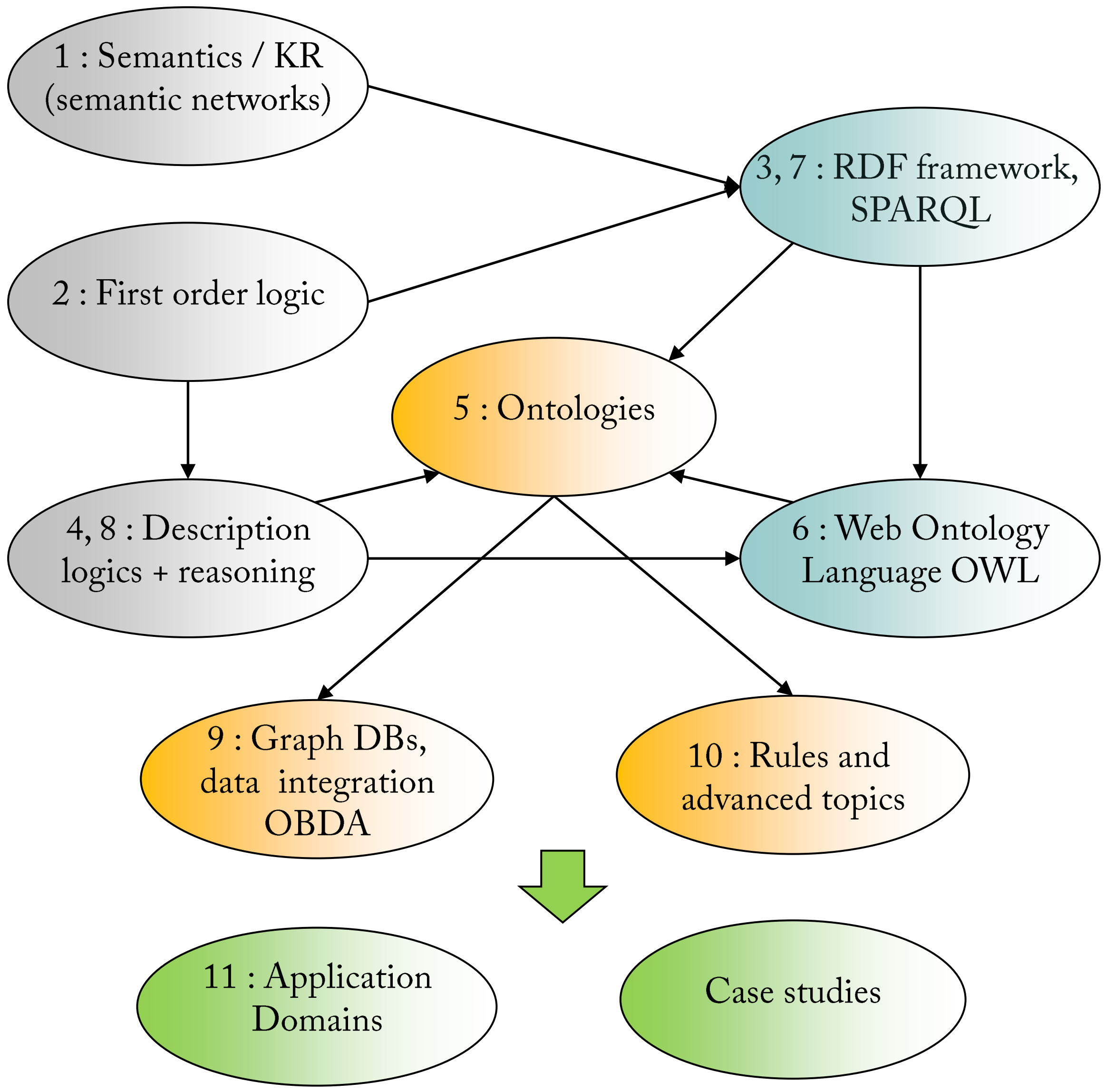
Course content outline

Credits : 5 (theory 25 h, practice 10 h, project 45 h)

Theory (25 h):

1. Semantics and knowledge representation.
2. Introduction to first order logic.
3. The semantic web resource description framework.
4. Description logics.
5. Ontologies and ontology engineering.
6. The Web Ontology Language : OWL.
7. Querying the semantic web : SPARQL.
8. Reasoning with description logics.
9. Data integration and ontology-based data access.
10. Rules and advanced topics.
11. Application domains for semantic data.

Case studies : real cases for genuine business customers; integrated in the relevant theory sessions.



Sources and recommended readings

- There are no additional required references for this chapter.

- Sources and useful additional readings :
  - *Artificial Intelligence: A Modern Approach (Russel and Norvig 2010).*
  - Chapter 1 from *Handbook of Knowledge Representation (van Harmelen et al. 2007).*

- University courses having partially inspired ideas and examples for this chapter :
  - *Propositional logic* and *First order logic, M. Frade, university of Minho, Portugal.*
  - *Introduction to first order logic, C. Ghidini and L. Serafini, Fondazione Bruno Kessler, Italy.*

Why study logic?

"Concepts and methods of logic occupy a central place in computer science, insomuch that logic has been called "the calculus of computer science." (Halpern and Harper 2001).

- Boolean algebra is widely used in programming languages.
- SAT Solvers (propositional logic SATisfiability solvers) solve large constraint problems in many domains : planning, diagnostic, design or model checking, PC configuration…
- First order logic has strong links with the relational database model.
- Logic is used for knowledge representation, knowledge agents and expert systems.
- Prolog is a programming language based on logic, used in AI and other areas.
- The semantic web standards rely on description logics.
- Program specifications and proofs of program correctness rely on logical formalisms.
- …

Types of logics

- Proposition logic (PL) : truth value of propositions.
  - For most students, the only type of logic seen so far; a very brief summary is provided in this chapter.
- First order predicate logic (FOL) : predicates, objects and functions.
  - Seen in this chapter (without equality). Reasoning algorithms will be kept for description logics.
- Description logics (DL) : structured descriptions of concepts.
  - Main formal basis of this course; seen in chapter 4 (presentation) and 8 (reasoning algorithms).
- Other logics :
  - Modal logic : apply modalities (*necessarily*, *possibly;* or *believes*, *knows* …) to propositions.
  - Temporal logic (a type of modal logic) : introduce temporal operators (*always*, *eventually* …).
  - Default logics : one of the main formal approaches to non-monotonic reasoning.
  - Fuzzy logic : degrees of truth (in the range [0, 1]) to express vagueness of propositions…

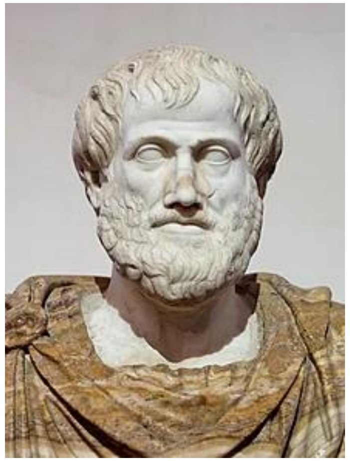
Focus point

- Our focus point is not the mathematical logical notation itself (although precise logical notations are used in this chapter).
- Our focus point is at the knowledge level :
  - What does the representation mean ? Can this be defined by declarative semantics ?
  - What reasoning processes / inferences are supported ?
  - What are their limitations ?

Agenda

1	First order logic
2	Model-theoretic semantics
3	Satisfiability, validity, entailment
4	Decidability

Precursors of first order logic



- The first systematic study of logic goes back to Aristotle (384 – 322 BC).
  - It was part of his body of works called the *Organon*.
  - He built a theory of syllogisms (deductions based on premises and a conclusion).

Among the founding fathers of modern predicate calculus :

- Gottlob Frege (1848, 1925).
- Charles Sanders Peirce (1839, 1914).

A very short reminder of proposition logic

- A logical proposition is either *True* or *False*. No other value is possible!
  - *Belgium is a Kingdom.* (a proposition considered as *True* in our real world)
  - *There exists a man who is immortal.* (a proposition considered as *False* in our real world)
  - *Please study !* (not a proposition)

- The vocabulary of propositional logic (PL) is made of :
  - Logical constants : true (also noted ⊤) and false (also noted ⊥) (*).
  - Propositional variables : a countably infinite set of variables $\Pi = \{p, p_1 \dots p_n, q, r \dots\}$ called atomic formulas.
  - Logical connectives : ¬ (negation), ∧ (conjunction), ∨ (disjunction), → (implication), ↔ (equivalence)

* : true and false, constants of the logical language, are not the same as truth values (noted *True* and *False)*, which characterize states in the real world. However, the distinction between them is often neglected in practice.

A very short reminder of proposition logic ./.

- Syntax (Backus-Naur definition) :

<*formula*>	::=	p | …	(for each p ∈ Π)
<*formula*>	::=	*true* | *false*	(also noted ⊤ and ⊥)
<*formula*>	::=	¬<*formula*>	
<*formula*>	::=	(<*formula*> <*connective*> <*formula*>)	([], {} can also be used)
<*connective*>	::=	∧ | ∨ | → | ↔	

Precedence of operators (higher precedence from left to right) : ¬ , ∧, ∨, →, ↔.

Examples : $a \vee b$ $c \rightarrow d \wedge e$ $p \leftrightarrow \neg(r \vee s)$

- The semantics of the logical connectives can be captured by truth tables :

a	b	¬(a)	∧ (a, b)	∨ (a, b)	→ (a, b)	↔ (a, b)
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Some standard equivalences in propositional logic

$(a \land b) \equiv (b \land a)$	commutativity of $\land$
$(a \lor b) \equiv (b \lor a)$	commutativity of $\lor$
$((a \land b) \land c) \equiv (a \land (b \land c))$	associativity of $\land$
$((a \lor b) \lor c) \equiv (a \lor (b \lor c))$	associativity of $\lor$
$\neg(\neg a) \equiv a$	double-negation elimination
$(a \rightarrow b) \equiv (\neg b \rightarrow \neg a)$	contraposition
$(a \rightarrow b) \equiv (\neg a \lor b)$	implication elimination
$(a \leftrightarrow b) \equiv ((a \rightarrow b) \land (b \rightarrow a))$	biconditional elimination
$\neg(a \land b) \equiv (\neg a \lor \neg b)$	De Morgan's laws
$\neg(a \lor b) \equiv (\neg a \land \neg b)$	De Morgan's laws
$(a \land (b \lor c)) \equiv ((a \land b) \lor (a \land c))$	distributivity of $\land$ over $\lor$
$(a \lor (b \land c)) \equiv ((a \lor b) \land (a \lor c))$	distributivity of $\lor$ over $\land$

These formulas should be known !

Limits of proposition logic

Proposition logic is not suited to express propositions of a parametric nature.
In first order logic, new constructs give us richer expressive possibilities.

- How to express that several propositions concern the same individual ?
  - *Michel is a Belgian and Michel is married.*

⇨
- Predicates and constants.

Belgian(Michel)
Married(Michel)

- How to express generalizations ?
  - *All humans are animals.*
  - *There exists at least one bird which cannot fly.*

⇨
- Variables, universal and existential quantifiers.

$\forall x\ (Human(x) \rightarrow Animal(x))$
$\exists x\ (Bird(x) \wedge Cannotfly(x))$

- How to express functional relations ?
  - *The wife of Michel is dead.*

⇨
- Functions.

Dead(wife(Michel))

Vocabulary of first order logic

- Non logical symbols : FOL allows us to talk about :
  - Constant symbols, representing objects : Jean, House21, 38 …
  - Predicate(arity) symbols, representing relations : Red(1), Prime(1), BrotherOf(2), PartOf(2) …
  - Function(arity) symbols : fatherOf(1), oneMoreThan(1), plus(2) …

  We will note constant and predicates with an uppercase first letter, functions with a lower case.

- Logical symbols :
  - Variables : x, y, z …
  - Connectives : $\neg, \rightarrow, \wedge, \vee, \leftrightarrow$
  - Quantifiers :

    $\forall$: universal (intuitively : the quantified formula is *True* for all values of the quantified variable).

    $\exists$: existential (intuitively : the quantified formula is *True* for at least 1 value of the quantified variable).

Syntax of first order logic

The language of first order logic includes two main elements :

- Terms represent objects (intuitively correspond to natural language noun phrases).
  - Terms can be constants or variables, or can be built by applying a function to other terms :

    Jean, Michel, 28, x, father(Michel)

- Formulas have a truth value (intuitively correspond to natural language sentences).
  - Atomic formulas can be built by applying predicates to terms :

    Person(Michel), Dead(father(Michel)), Married(Michel, Sarah)

  - Complex formulas can be built by using same connectives as in propositional logic :

    $Dead(Michel) \vee Alive(Michel)$

  - Complex formulas can also be built by applying quantifiers to variables :

    $\forall x\ (Human(x) \rightarrow Animal(x)),\ \exists x\ (Bird(x) \wedge Cannotfly(x))$

First order logic: formal syntax

- Terms are defined recursively as follows :
  - An individual constant or a variable is an atomic term.
  - If f is a functional symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is also a term.

- Formulas are defined recursively as follows :

  If F and G are formulas, P a predicate, $t_1, \ldots, t_n$ are terms, and x a variable then :
  - $P(t_1, \ldots, t_n)$ is an atomic formula.
  - true (also noted $\top$) and false (also noted $\bot$) are formulas.
  - $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$ are formulas.
  - $\forall x\ F$ and $\exists x\ F$ are formulas.

Order of higher precedence from left to right is : $\forall$ and $\exists$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.

We accept to omit parentheses where possible, using precedence of operators.

We may also use [ ] and { } to express parentheses when it makes the notation clearer.

Examples of terms and formulas

- **Terms**

an individual named Michel	*Michel*
the number 1	*1*
the wife of Michel	*wife(Michel)*
anything	x
the # of victories of McEnroe against Borg	*victories(McEnroe, Borg)*
the # of victories of McEnroe's brother against Borg	*victories((brother(McEnroe), Borg)*

- **Formulas**

Paris is a city and France is a country.	$City(Paris) \land Country(France)$
Paris is in France.	$Location(Paris, France)$
There is at least one city in France.	$\exists x\ (City(x) \land Location(x, France))$

Intuitive understanding of quantifiers and common mistakes

- $\forall$ means for all.
  - The sentence $\forall x\ F$, where F is a logical formula, says that F is *True* for every object x.
  - We can use it to make universal statements such as *every man is mortal.*

  $\forall x\ (Human(x) \rightarrow Mortal(x))$

⚠ $\forall x\ (Human(x) \land Mortal(x))$ has another meaning : all objects are human and all objects are mortal.

- $\exists$ means there exists.
  - The sentence $\exists x\ F$, where F is a logical formula, says that F is *True* for at least one object x.
  - We can use it to make existential statements such as *there is at least one red Ferrari*.

  $\exists x\ (Red(x) \land Ferrari(x))$

⚠ $\exists x\ (Red(x) \rightarrow Ferrari(x))$ has another meaning : there is one object such as, if it is red, it is a Ferrari.

Universal quantification is typically used with implication, existential with conjunction.

Nested quantifiers

- Expressing more complex sentences will often require several quantifiers.

- If they are of the same type, the order of quantifiers is not important :

  Brothers are siblings. $\forall x\ \forall y\ (Brother(x, y) \rightarrow Sibling(x, y))$

- If they are of mixed types, the order of quantifiers influences the meaning :

  Everybody loves somebody. $\forall x\ \exists y\ Loves(x, y)$

  There is someone who is loved by everyone. $\exists y\ \forall x\ Loves(x, y)$

De Morgan's rules for quantifiers

- There is a relationship between universal and existential quantifiers.
- $\forall$ is a conjunction over the universe of objects while $\exists$ is a disjunction.

  As a consequence, they obey De Morgan's rules :

De Morgan's rules without quantifiers:	Generalized de Morgan's rules:
$\neg(P \lor Q) \leftrightarrow \neg P \land \neg Q$	$\forall x\ \neg F \leftrightarrow \neg\exists x\ F$
$\neg(P \land Q) \leftrightarrow \neg P \lor \neg Q$	$\neg\forall x\ F \leftrightarrow \exists x\ \neg F$
$P \land Q \leftrightarrow \neg(\neg P \lor \neg Q)$	$\forall x\ F \leftrightarrow \neg\exists x\ \neg F$
$P \lor Q \leftrightarrow \neg(\neg P \land \neg Q)$	$\exists x\ F \leftrightarrow \neg\forall x\ \neg F$

- We could use only one quantifier, but for readability reasons we will keep both.
- Similarly, we could use only conjunction or disjunction, but we will keep both.

Free and bound variables, quantifier scope

- The scope of a quantification is the formula to which the quantification is applied :

  In the expressions $\forall x\ F$ or $\exists x\ F$, the scope of x is the formula F.

  - Any occurrence of a variable x in the scope of a quantification $\forall x$ or $\exists x$ is bound.
  - The variable x in the quantification $\forall x$ or $\exists x$ is said to be quantified.
  - A variable which is not bound nor quantified is said to be free.

- If a formula has free variables, its truth value depends on their values.

  In $\forall x\ P(x, y)$, x is bound by the universal quantifier $\forall$, but y is free.

  - In order to interpret the formula, we need to assign a value to the free variable.

  If P is the relation $\geq$ and the domain is the set of naturals, the formula is *True* if $y = 0$, but *False* if $y = 2$.

Caution note

- A variable is bound by the innermost quantifier mentioning it :

$$\forall x\ (\text{Country}(x) \lor \exists x\ \text{Location}(\text{Paris}, x))$$

is equivalent to and should be written as :

$$\forall x\ (\text{Country}(x) \lor \exists y\ \text{Location}(\text{Paris}, y))$$

- Using different variables with nested quantifiers is less confusing.

Why is it called first order logic ?

- The name indicates that in FOL we may quantify over individual variables.
- In higher logics we would be able to quantify over other types of elements.
  - For example over predicates :

    If P is a predicate symbol, $\exists P\ P(b)$ is not a sentence of FOL.

    But this is a legitimate sentence of second-order logic.

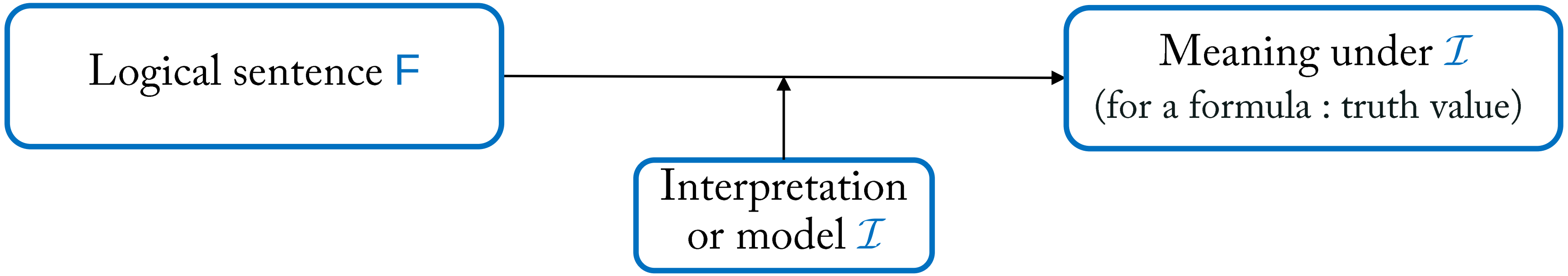
Agenda

1. First order logic
2. **Model-theoretic semantics**
3. Satisfiability, validity, entailment
4. Decidability

Model theoretic semantics

- In the *correspondence theory of truth*, the meaning of logical formulas should be assessed by correspondence to the real world. That is however far too complex.

- Model-theoretic semantics use models :
  - A model is a description of a small "possible world", "state of affairs", or fragment of reality.
  - A model specifies information which allows to interpret a sentence and find out its meaning.



- If S is a sentence of the language, $S^{\mathcal{I}}$ represents the meaning of S under interpretation $\mathcal{I}$.

- An interpretation or model $\mathcal{I}$ satisfies a sentence F or is a model of F if $F^{\mathcal{I}}$ is *True*.

  In that case we will note it as $\mathcal{I} \models F$.

Ingredients of model-theoretic semantics

Model-theoretic semantics apply to many logics, with the same ingredients :

1. A definition of how to build a model;

2. A definition a priori of the meaning of atomic syntactic structures w.r.t. that model;

   That definition is provided by the interpretation function $.^{\mathcal{I}}$.

3. A formal (recursive) definition of the meaning of complex syntactic structures in terms of the atomic ones using the constructors of the language.

The specific details depend on the logic being considered.

Model theoretic semantics of propositional logic (reminder)

- Propositional logic only deals with dependencies between the truth values of statements.
- A PL model or interpretation $\mathcal{I}$ is an assignment from propositional variables into truth values.
  - Example : for the variables p, q and r, $\mathcal{I} = \{p = True; q = True; r = False\}$ is a possible model.
- For any interpretation $\mathcal{I}$ and any formulas F and G, the truth value assigned by $\mathcal{I}$ is determined recursively by the following rules :

  - The truth value of any propositional variable $p^{\mathcal{I}}$ is directly specified in $\mathcal{I}$.
  - $\top^{\mathcal{I}} = True$; $\bot^{\mathcal{I}} = False$.
  - $(\neg F)^{\mathcal{I}}$ is *True* iff $F^{\mathcal{I}}$ is *False*.
  - $(F \land G)^{\mathcal{I}}$ is *True* iff $F^{\mathcal{I}}$ is *True* and $G^{\mathcal{I}}$ is *True*.
  - $(F \lor G)^{\mathcal{I}}$ is *True* iff either $F^{\mathcal{I}}$ is *True* or $G^{\mathcal{I}}$ is *True*.
  - $(F \rightarrow G)^{\mathcal{I}}$ is *True* unless $F^{\mathcal{I}}$ is *True* and $G^{\mathcal{I}}$ is *False*.
  - $(F \leftrightarrow G)^{\mathcal{I}}$ is *True* iff $F^{\mathcal{I}}$ and $G^{\mathcal{I}}$ are both *True* or both *False*.

Semantics of FOL: intuition

- We will build a definition of FOL semantics by describing how an interpretation or model maps expressions of the language to the real world.
- FOL formulas can be *True* or *False*; semantics must define their truth value.
- However FOL has additional language elements, including objects.
- The interpretation domain is the set of objects referred by an interpretation.

Using that interpretation domain, we will build correspondences between :

- *constant symbols* and **objects** of the interpretation domain;
- *predicate symbols* and **relations** (sets of n-tuples) on the interpretation domain;
- *function symbols* and **functions** on the interpretation domain.

Interpretation domain : example

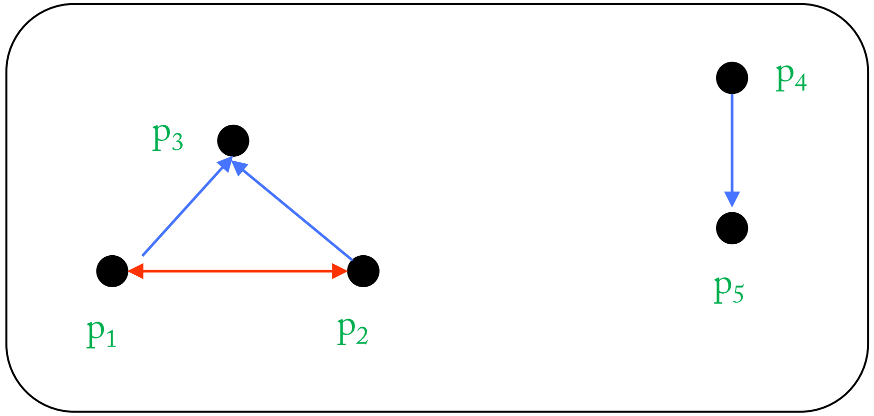
- KB : constants *Jean*, *René*, *Alice*, *Michel*, *Sarah*, a function *manager* and a predicate *Work_with*.

- A possible interpretation $\mathcal{I}_1$ is:
  - Jean$^{\mathcal{I}_1}$ = p_1; René$^{\mathcal{I}_1}$ = p_2; Alice$^{\mathcal{I}_1}$ = p_3; Michel$^{\mathcal{I}_1}$ = p_4; Sarah$^{\mathcal{I}_1}$ = p_5.
  - manager$^{\mathcal{I}_1}$: {p_1 -> p_3; p_2 -> p_3; p_4 -> p_5}
  - Works_with$^{\mathcal{I}_1}$: *True* for {<p_1, p_2>}

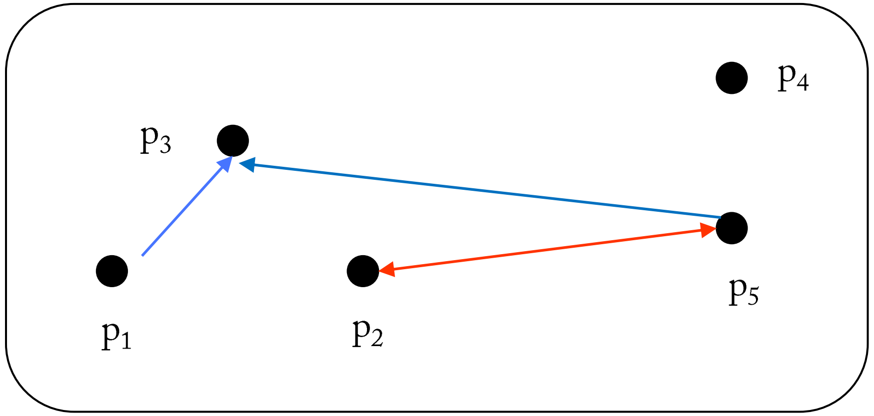
- Another model with the same domain.

  Works_with(*Jean*, *René*) is *True* under $\mathcal{I}_1$, but not under $\mathcal{I}_2$.

Interpretation domain and model $\mathcal{I}_1$



Interpretation domain and model $\mathcal{I}_2$



First order logic semantics

A first order interpretation $\mathcal{I}$ is a triple $\{ \Delta^{\mathcal{I}}, .^{\mathcal{I}}, \alpha \}$, where

- ❑ $\Delta^{\mathcal{I}}$ is a non-empty set called the interpretation domain;
- ❑ $.^{\mathcal{I}}$ is an interpretation function associating :
  - to every constant symbol c_i an element of the domain $c_i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
  - to every predicate symbol P_i of arity n a relation $P_i^{\mathcal{I}} : \Delta^{\mathcal{I}\,n} \rightarrow \{True, False\}$ (*),
  - to every function symbol f_i of arity m a function $f_i^{\mathcal{I}} : \Delta^{\mathcal{I}\,m} \rightarrow \Delta^{\mathcal{I}}$.
- ❑ α is an assignment associating to each free variable x_i an element of the domain : $\alpha(x_i) \in \Delta^{\mathcal{I}}$.
- ❑ We will note $t^{\mathcal{I}}$ and $F^{\mathcal{I}}$ the meaning assigned to any term t or formula F by $\mathcal{I}$.

* : the notation $\Delta^{\mathcal{I}\,n}$ denotes the n-ary cartesian product of $\Delta^{\mathcal{I}}$.

First order logic semantics : terms

The interpretation function $.^{\mathcal{I}}$ is defined recursively.

- For terms :
  - For a constant c, its interpretation is given directly by the interpretation function $c^{\mathcal{I}}$.
  - For a free variable x, its interpretation is given by the variable assignment $\alpha : x^{\mathcal{I}} = \alpha(x)$.
  - For a functional symbol f of arity m, if $t_1, t_2 \ldots t_m$ are terms : $f(t_1, t_2 \ldots t_m)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, t_2^{\mathcal{I}} \ldots t_m^{\mathcal{I}})$.

First order logic semantics : formulas

- For atomic formulas :
  - If P is a predicate symbol of arity n, and if $t_1, t_2 \ldots t_n$ are terms : $P(t_1, t_2 \ldots t_n)^{\mathcal{I}} = P^{\mathcal{I}}(t_1^{\mathcal{I}}, t_2^{\mathcal{I}} \ldots t_n^{\mathcal{I}})$.
- For logical constants and formulas with connectives ($\neg, \rightarrow, \wedge, \vee, \leftrightarrow$) :
  - Their interpretation follows the same rules as in propositional logic.
- For formulas with quantifiers :

  Let us note $\mathcal{I}_{x=d}$ the variant of interpretation $\mathcal{I}$ where free variable x has value d ($d \in \Delta^{\mathcal{I}}$). Then :
  - $(\forall x\ F)^{\mathcal{I}} = \mathit{True}$ iff for all elements $d \in \Delta^{\mathcal{I}}$, $F^{\mathcal{I}_{x=d}} = \mathit{True}$.
  - $(\exists x\ F)^{\mathcal{I}} = \mathit{True}$ iff there is exists one element $d \in \Delta^{\mathcal{I}}$ such as $F^{\mathcal{I}_{x=d}} = \mathit{True}$.

Agenda

Fundamental semantic concepts

We will use model-theoretic semantics to define key concepts usable in any logic :

- Satisfiability.
- Validity.
- Logical equivalence.
- Entailment, or logical consequence.

Satisfiability and validity

- An interpretation or model $\mathcal{I}$ satisfies a formula F, noted $\mathcal{I} \models F$, if $F^{\mathcal{I}}$ is *True*.
  - In that case we say that $\mathcal{I}$ is a model of F.
  - If F is *False* under interpretation $\mathcal{I}$, we say that $\mathcal{I}$ does not satisfy F, noted $\mathcal{I} \not\models F$.
- A formula F is satisfiable if it has at least one model.
- A formula F is valid, noted $\models F$, if it is *True* for all models.
- A formula F is unsatisfiable, noted $\not\models F$, if it is *False* under all models.
  - A formula is valid iff its negation is unsatisfiable : $\models F$ iff $\not\models \neg F$.

Examples :

$p \lor q$ is satisfiable.

$p \lor \neg p$ is valid.

$p \land \neg p$ is unsatisfiable.

Satisfiability for set of formulas

- An interpretation or model $\mathcal{I}$ satisfies a set of formulas E if it satisfies each formula in E.

- A set of formulas E is satisfiable if E has at least one model :

  there is at least one interpretation $\mathcal{I}$ satisfying all formulas in E.

- A set of formulas E is unsatisfiable if it is not possible to satisfy it :

  for every interpretation $\mathcal{I}$, we have at least one formula $F \in E$ where $I \not\models F$.

Consequence :

- The models of a finite set of formulas $\{F1, \dots, Fn\}$ are the same as the models of the unique formula being the conjunction $F1 \wedge \dots \wedge Fn$.

Entailment and logical equivalence

- A formula G entails a formula F, noted $G \models F$, if every model of G is a model of F (*).
  - We also say then that F is a logical consequence of G.

- A set of formulas E entails a formula F, or F is a logical consequence of E, noted $E \models F$, if every model of E is a model of F.
  - If E is empty, E can only entail F if F is valid : $\emptyset \models F$ is equivalent to $\models F$.

- Two formulas F and G are logically equivalent, noted $F \equiv G$, if they have the same models :

  $F^{\mathcal{I}} = G^{\mathcal{I}}$ for any interpretation $\mathcal{I}$.

* : The symbol $\models$ is used to represent both entailment and satisfiability, which are tightly connected.

Some useful results

For any sentences F and G :

- $G \models F$ iff $\models (G \rightarrow F)$: G entails F iff $G \rightarrow F$ is valid (deduction theorem).

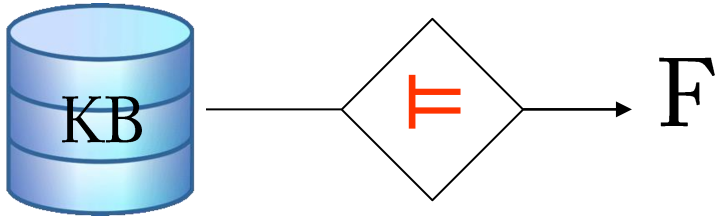
- $G \models F$ iff $\not\models (G \land \neg F)$: G entails F iff $G \land \neg F$ is unsatisfiable.

These results will be very useful when we discuss logical reasoners and decision procedures.

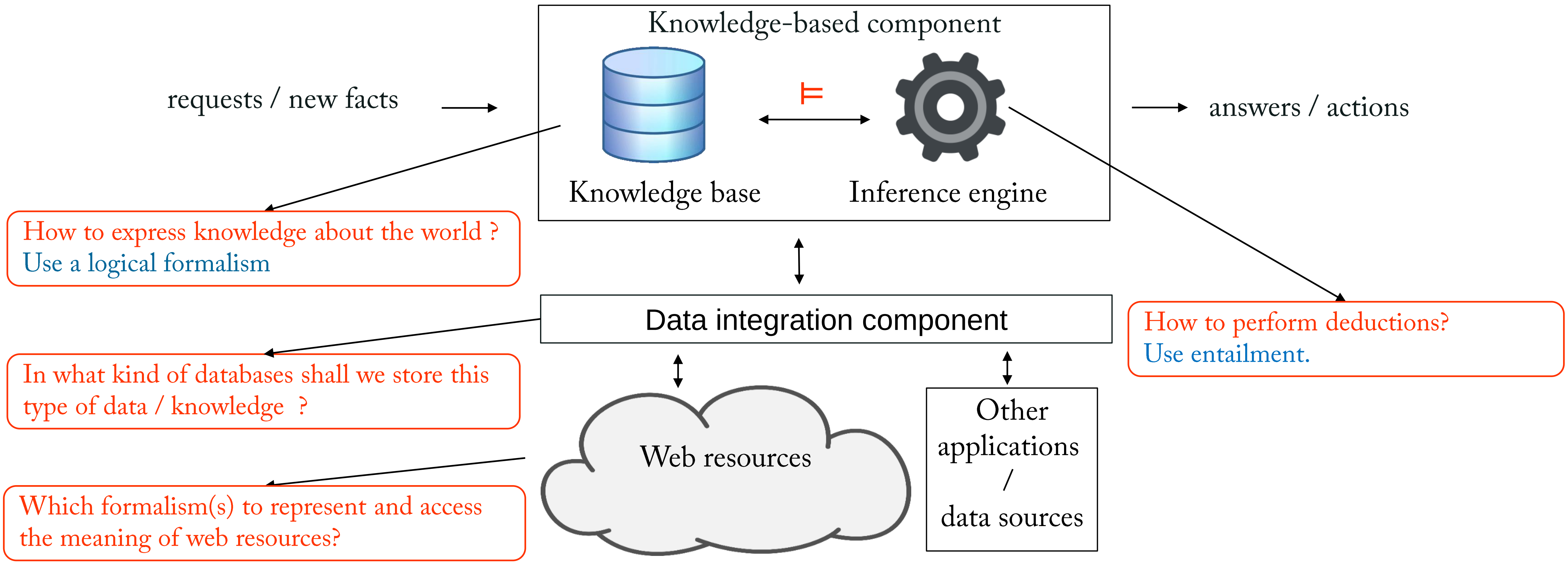
Logical consequence and knowledge-based inferences

- A knowledge base KB is a set of sentences expressed in a knowledge representation language.
  - A knowledge base using logic is a set of logical formulas.
  - It can be considered as a single formula being the conjunction of all formulas included in it.
- In a KB using logic, inference is related to entailment :

  A knowledge base KB allows to infer a formula F iff F is entailed by KB, or iff F is *True* in all models where KB is *True*.



Updated framework of reference



Monotonicity, closed-world and open-world assumptions

- **Monotonicity** : the set of entailed sentences can only increase as information is added to the KB.
  - Facts known as *True* remain *True* regardless of what is else is in the KB or what is added to it :

    For any sentences F and G, if $KB \vDash F$ then $KB \wedge G \vDash F$.
  - Default inheritance (cf. chapter 1) leads to non-monotonicity.

- **Closed-world assumption (CWA)** : sentences not known to be *True* are assumed *False*.
  - Database systems usually make the closed-world assumption.
  - CWA is associated with non-monotonicity : if a formula F is not in KB then $KB \vDash \neg F$, but $KB \wedge F \vDash F$.

- **Open-world assumption (OWA)** : sentences not known to be *True* or *False* are *unknown*.
  - Description logics and the semantic web make the open-world assumption, as the web can always provide more information.
  - OWA is associated with monotonicity.

Agenda

1	First order logic
2	Model-theoretic semantics
3	Satisfiability, validity, entailment
4	Decidability

Questions regarding the use of FOL for reasoning

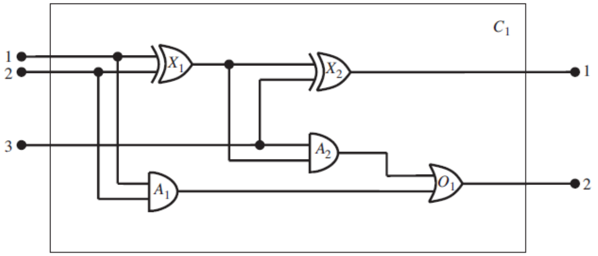
- Entailment is our inference mechanism of choice.
- The model-theoretic semantics on which it relies apply to many logics.
- Is first order logic our logical formalism of choice ?

We will look at this question from two points of view :

1. Suitability of the formalism for representing knowledge.
2. Capability to define efficient reasoning algorithms.

1. Knowledge representation in FOL

Excerpt from an example KB about circuits and gates *(source Russel and Norwig 2010)*



1. If two terminals are connected, they have the same signal.
2. The signal at every terminal is either 1 or 0.
3. Connected is commutative.
4. There are four types of gates.
5. An AND gate's output is 0 if and only if any of its inputs is 0.
6. An OR gate's output is 1 if and only if any of its inputs is 1.
7. An XOR gate's output is 1 if and only if its inputs are different.
8. Gates are circuits.

…

1. $\forall t1, \forall t2\ (Terminal\ (t1) \land Terminal\ (t2) \land Connected(t1, t2) \rightarrow Signal\ (t1)=Signal\ (t2))$
2. $\forall t\ (Terminal\ (t) \rightarrow Signal\ (t)=1 \lor Signal\ (t)=0)$
3. $\forall t1, \forall t2\ (Connected(t1, t2) \leftrightarrow Connected(t2, t1))$
4. $\forall g\ (Gate(g) \land k = Type(g) \rightarrow k = AND \lor k = OR \lor k = XOR \lor k = NOT)$
5. $\forall g\ (Gate(g) \land Type(g)=AND \rightarrow Signal\ (Out(1, g))= 0 \leftrightarrow \exists n\ Signal\ (In(n, g))=0)$
6. $\forall g\ (Gate(g) \land Type(g)=OR \rightarrow Signal\ (Out(1, g))= 1 \leftrightarrow \exists n\ Signal\ (In(n, g))=1)$
7. $\forall g\ Gate(g) \land Type(g)=XOR \Rightarrow Signal\ (Out(1, g))= 1 \Leftrightarrow Signal\ (In(1, g)) \neq Signal\ (In(2, g))$
8. $\forall g\ (Gate(g) \rightarrow Circuit(g))$

…

Encoding a particular circuit :

a. $Circuit(C1) \land Arity(C1, 3, 2)$

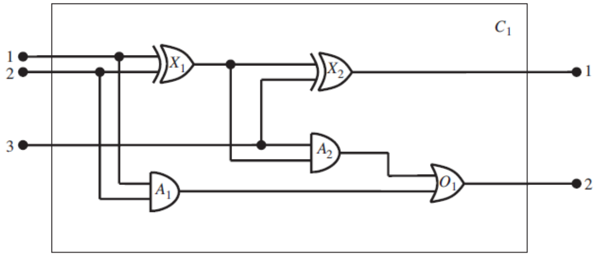
b. $Gate(X1) \land Type(X1)=XOR$

c. …

What to think about the use of FOL to encode knowledge ?

1. Knowledge representation in FOL

Excerpt from an example KB about circuits and gates *(source Russel and Norwig 2010)*



1. $\forall t1, \forall t2$ (Terminal (t1) ∧ Terminal (t2) ∧ Connected(t1, t2) ⟶ Signal (t1)=Signal (t2))
2. $\forall t$ (Terminal (t) ⟶ Signal (t)=1 ∨ Signal (t)=0)
3. $\forall t1, \forall t2$ (Connected(t1, t2) ↔ Connected(t2, t1))
4. $\forall g$ (Gate(g) ∧ k = Type(g) ⟶ k = AND ∨ k = OR ∨ k = XOR ∨ k = NOT)
5. $\forall g$ (Gate(g) ∧ Type(g)=AND ⟶ Signal (Out(1, g))= 0 ↔ ∃n Signal (In(n, g))=0)
6. $\forall g$ (Gate(g) ∧ Type(g)=OR ⟶ Signal (Out(1, g))= 1 ↔ ∃n Signal (In(n, g))=1)
7. $\forall g$ Gate(g) ∧ Type(g)=XOR ⇒ Signal (Out(1, g))= 1 ⇔ Signal (In(1, g)) ≠ Signal (In(2, g))
8. 12. $\forall g$ (Gate(g) ⟶ Circuit(g))

…

Encoding a particular circuit :

a. Circuit(C1) ∧ Arity(C1, 3, 2)
b. Gate(X1) ∧ Type(X1)=XOR
c. …

✓ It can be done.
✓ It supports precise axiomatization.

✗ Knowledge is not easy to structure and to manage in FOL:

- Where are the concept definitions ?
- Where is the hierarchy ?

FOL is a language of choice for describing axiomatized theories but not for practical KBs.

2. Logical reasoning

- A logical reasoner is an algorithm checking entailment.
- Let us note M(F) the set of all possible models of a formula F. Then :

  $KB \vDash F$ iff $M(KB) \subseteq M(F)$ by definition of entailment (note the sense of the inclusion).

- Entailment can be tested by a systematic model checking algorithm :
  - The algorithm will enumerate all possible models and checks if F is true whenever KB is true.
- This algorithm will only terminate if the number of possible models is finite.
- Is that the case in propositional logic?
  - Yes, why ?
  - If n is the number of propositional variables, there are 2^n of combinations of truth values.

A truth table enumeration algorithm for PL entailment

```
function TT-ENTAILS?(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   symbols ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, { })
```

```
function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true // when KB is false, always return true
   else do
      P ← FIRST(symbols)
      rest ← REST(symbols)
      return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
              and
              TT-CHECK-ALL(KB, α, rest, model ∪ {P = false }))
```

Figure 7.10 A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns *true* if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword "**and**" is used here as a logical operation on its two arguments, returning *true* or *false*.

(Source Russel and Norvig 2010)

- This algorithm performs a recursive enumeration on a finite space of assignments to propositional symbols.
- It is sound : it only derives formulas entailed by the KB.
- It is complete : it can derive any formula entailed by KB and always terminate.
- Its time complexity is $O(2^{n})$, where n is the number of propositional symbols, and space complexity is $O^{(n)}$.
- More efficient algorithms exist, but propositional satisfiability is NP-complete (Cook's theorem).

Every known inference algorithm for propositional logic has a worst-case complexity probably exponential in the size of the input.

Quid of FOL ?

Decision problems

- A decision problem is any problem that, given a certain input, asks a question to be answered with “yes” or “no”.
  - Entailment checking is an example of a decision problem.
- A problem is decidable if it can always receive a correct “yes” or “no” answer for every instance of the input.
- In formal logic we have several important decision problems :
  - Validity problem: Is F valid ?
  - Satisfiability problem: Is F satisfiable ?
  - Entailment or logical consequence problem: Is G a consequence of F ?
  - Equivalence problem: Are F and G equivalent ?
- Are these questions decidable ?

Decidability of propositional logic

- We have seen that:

▪ $F \models G$	iff	$\not\models (G \land \neg F)$: F entails G iff $G \land \neg F$ is not satisfiable.
▪ $\models F$	iff	$\not\models \neg F$: F is valid iff $\neg F$ is unsatisfiable.
▪ F is satisfiable	iff	$\neg F$ is not valid.
▪ $F \equiv G$	iff	$F \models G$ and $G \models F$.

Any algorithm that works for one of our decision problems will work for all !

- Are the above problems in propositional logic decidable ?

- Yes ! For PL, truth table enumeration is guarantied to terminate in finite time.

=> Propositional logic is decidable, and its worst-case time complexity is $O(2^n)$.

- PL decision algorithms, called SAT solvers, have focused on the SATisfiability problem.
- In practice, truth-table enumeration is not used. The best algorithms use backtracking search.

Fundamental complexity is unchanged, but optimizations allow to handle tens of millions of variables.

Undecidability of first order logic

- We consider the same decision problems :
  - Validity problem: is F valid ?
  - Satisfiability problem: is F satisfiable ?
  - Entailment or logical consequence problem: is G a consequence of F ?
  - Equivalence problem: are F and G equivalent ?
- Alan Turing (1936) and Alonzo Church (1936) both proved independently that the questions of validity and entailment for first-order logic are semi-decidable.
  - Given the question $F \models G$?, algorithms exist that say yes to every entailed sentence, but they are not guarantied to terminate in the case of non-entailment. No algorithm will always answer by yes or no.
- The question of satisfiability is not even semi-decidable : it is undecidable.
  - If it was semi-decidable, we could answer on the satisfiability of $\neg F$ and hence on the non validity of F.

Practical approaches for reasoning with first order logic

- Two important approaches have focused on specific subsets of first order logic :

1. Definite clauses => Prolog.
   - Definite clauses : disjunctions of literals of which exactly one is positive.

   $\neg P(X, Y) \vee \neg Q(Y, Z) \vee R(X, Z)$, which is equivalent to $P(X, Y) \wedge Q(Y, Z) \rightarrow R(X, Z)$

   Definite clauses in FOL are still undecidable but lend themselves to fast implementations.

2. Description Logics => Semantic web, OWL.
   - Description logics work with decidable fragments of first order logic.
   - They are the theoretical basis for this course.

Summary

- Proposition logic is decidable and has very useful and efficient reasoning algorithms for specialized tasks but is not expressive enough for most knowledge-based systems.
- First order logic is a powerful knowledge representation language, the standard to which other languages are compared. It presents however two drawbacks :
  - FOL is **too expressive,** to the extend that FOL is not decidable.
  - FOL is not **structured** for knowledge representation : it lacks the syntactic tools to organize knowledge, identify concept definitions, make visible the taxonomic hierarchy…
- Some restricted fragments of first order logic are decidable and support efficient reasoners; they will offer a solution for our purpose.

References

- *[Halpern and Harper 2001] : Halpern J. and Harper R.*, On the Unusual Effectiveness of Logic in Computer Science, *The Bulletin of Symbolic Logic, 7, 2, 2001.*

- *[Russel and Norvig 2010]: Russel S. and Norvig, P.:* Artificial Intelligence: A Modern Approach (3rd Edition), *Pearson, 2010.*

- *[van Harmelen et al. 2007]: van Harmelen F., Lifschitz V. and Porter B., Eds.*, Foundations of Artificial Intelligence, *Volume 3 : Handbook of Knowledge Representation, Elsevier, 2007.*

THANK YOU