KRR2022 - Week 1: Propositional logic Course

Pierre Mercuriali
Responsible instructor: Carlos Hernandez Corbato

Spring 2022

What is (formal) logic? How does it help us roboticians?

The chicken is ready to eat.
I saw a man on a hill with a telescope.
He fed her cat food.

How to communicate with people, robots... without ambiguity? How to clearly state what we know or do not know, as well as our goals?

$$\frac{\text{Socrates is a man} \qquad \text{all men are mortal}}{\textit{Therefore}\ \text{Socrates is mortal}}$$

How to generalise this "reasoning structure"?

Short history

- With Aristotle, logic was conceived as a study of such syntactic "reasoning structures" (syllogisms).
- Very early, the idea appears of creating artificial *formal languages* for logical investigations.
- With Leibniz, the inspiration becomes mathematic, centered around *symbolic manipulation*.

  "The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right. " (The Art of Discovery, 1685)

- Afterwards, Frege, Peano, Russell, etc. devise formal notations and rules to work with them: modern formal logic is born.

Propositional logic: overview

Propositional logic: syntax and semantics

Proof systems: theorem proving and resolution, model checking

Horn clauses, forward and backward chaining
- Horn clauses
- SLD resolution

Computability and complexity
- Computability
- Complexity

Propositional logic

Propositional logic is one of the simplest logics, yet it is powerful enough for many knowledge-based applications. It is the starting point for logics that are even more powerful in terms of description: first-order logic, description logics...

Syllabus

Syllabus:

- Russell SJ, Norvig P. *Artificial Intelligence: A Modern Approach.* (Fourth edition, 2021)
- Ertel W, Mast F. *Introduction to Artificial Intelligence.* (Second edition, 2017)

What happens if it rains?

- In propositional logic (PL), we deal with *propositions* separated by *logical operators*.
- It makes it easier to describe and manipulate concepts.

Example

The sentence in natural language

> *"If it is raining, then I take my umbrella" / "When it is raining, then I take my umbrella"*

can be expressed using the symbol $\Rightarrow$ that denotes the *implication*:

> *It is raining* $\Rightarrow$ *I take my umbrella*

or even more concisely:

> *raining* $\Rightarrow$ *umbrella.*

Syntax: how to build a formula?

We adopt the formalism of *Introduction to Artificial Intelligence*.
To build formulas, we begin with smallest building blocks (*atoms*) and structure them together using logical operators.

Definition

Let $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$ be the set of *logical operators* and Σ a set of *symbols*. The sets $Op, \Sigma, \{t, f\}$ are pairwise disjoints. Σ is called the *signature* and its elements are called *propositional variables*. The set of *propositional formulas* is inductively (recursively) defined as follows:

- t and f are (atomic) formulas;
- all propositional variables are (atomic) formulas;
- if A and B are formulas, then $\neg A$, (A), $A \wedge B$, $A \vee B$, $A \Rightarrow B$, and $A \Leftrightarrow B$ are formulas.

Example

Example

The expression

$$raining \Rightarrow umbrella$$

is a formula in PL, where $\{raining, umbrella\}$ is a set of propositional variables. The expression

$$(raining \land \neg\, umbrella) \Rightarrow wet$$

is also a formula in PL. As en exercise, try to reformulate what it means in natural language (several answers are possible).

How to read a formula?

Definition

We read the symbols and operators in the following way:

t	: "true"	
f	: "false"	
$\neg A$	: "not A"	(negation)
$A \wedge B$	: "A and B"	(conjunction)
$A \vee B$	: "A or B"	(disjunction)
$A \Rightarrow B$	: "A implies B" or "if A, then B"	(implication)
$A \Leftrightarrow B$	: "A if and only if B"	(equivalence)

Towards semantics?

- We now know how to build formulas "mechanically".
- How do we add *meaning* to those syntactic expressions?

Semantics: how to give meaning to a formula?

- There are two truth values in PL: t for "true", f for "false".
- To know whether a *formula* is true or false, we give meaning to atoms and operators.

Definition

A mapping $I : \Sigma \rightarrow \{t, f\}$, which assigns a truth value to every propositional variable, is called an *interpretation*.

Example

To compute the truth value of the formula (rainy $\wedge$ sunny), we need to know the truth values of both atoms "rainy" and "sunny", and how $\wedge$ behaves: for instance, if both "rainy" and "sunny" are both true, then "(rainy $\wedge$ sunny)" is true.

Truth tables: defining logical operators

One way to define our basic operations is to describe exhaustively their behaviour. To this end, we range over every possible interpretation in what is called a *truth table*.

Example

To define the negation $\neg$, we give the value of $\neg A$ for every possible value of A: if A is true, then $\neg A$ is false; if A is false, then $\neg A$ is true.

Definition

A	B	(A)	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
t	t	t	f	t	t	t	t
t	f	t	f	f	t	f	f
f	t	f	t	f	t	t	f
f	f	f	t	f	f	t	t

Some vocabulary

Definition

Two formulas F and G are called *equivalent* if they take on the same truth value for all interpretations. We write $F \equiv G$.

Example

The symbol for equivalence can be used to define operators or express some of their properties. For instance, the *symmetry*, or *commutativity*, of the conjunction is expressed by $A \wedge B \equiv B \wedge A$.

Some vocabulary

Definition

A formula is called

- *satisfiable* if it is true for at least one interpretation;
- *valid* if it is true for all interpretations;
- *unsatisfiable* if it is not true for any interpretation.

Definition

A *model* for a formula is an interpretation that makes the formula true.

Example

The formula "soft_grip $\wedge$ strong_grip" is satisfiable: a model for it is $\{$soft_grip $\leftarrow t,$ strong_grip $\leftarrow t\}$. However, the formula "soft_grip $\wedge$ f" is unsatisfiable.

Useful equivalences

$A \land B \equiv B \land A$	symmetry	(1)
$A \lor B \equiv B \lor A$	symmetry	(2)
$A \land (B \land C) \equiv (A \land B) \land C$	associativity	(3)
$A \lor (B \lor C) \equiv (A \lor B) \lor C$	associativity	(4)
$A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$	distributivity	(5)
$A \land (B \lor C) \equiv (A \land B) \lor (A \land C)$	distributivity	(6)
$\neg\neg A \equiv A$		(7)
$\neg(A \land B) \equiv \neg A \lor \neg B$	de Morgan's laws	(8)
$\neg(A \lor B) \equiv \neg A \land \neg B$	de Morgan's laws	(9)
$A \Rightarrow B \equiv \neg B \Rightarrow \neg A$		(10)
$A \Rightarrow B \equiv \neg A \lor B$	implication	(11)
$A \Leftrightarrow B \equiv (A \Rightarrow B) \land (B \Rightarrow A)$		(12)
$A \oplus B \equiv \neg(A \Leftrightarrow B)$		(13)

Useful equivalences involving the constants

$$A \wedge f \equiv f \tag{14}$$
$$A \wedge t \equiv A \tag{15}$$
$$A \vee f \equiv A \tag{16}$$
$$A \vee t \equiv t. \tag{17}$$

These can be easily verified with a truth table.

Min and max

Remark that $\wedge$ and $\vee$ mimic the behaviour of the functions min and max, respectively, over the subset of the reals numbers $[0, 1]$ and with 0 and 1 corresponding to f and t, respectively:

$$\begin{aligned} \forall x \in [0,1], \quad \min(x,0) &= 0 \\ \min(x,1) &= x \\ \max(x,0) &= x \\ \max(x,1) &= 1. \end{aligned}$$

This is useful to remember how $\wedge$ and $\vee$ behave. It is also useful in multi-valued and fuzzy logic, where the domain of the interpretation can be real, natural numbers, or, more generally, a partially ordered set.

Motivation

In AI, we are interested in taking existing knowledge and, from that,

- deriving new knowledge,
- answering queries.

Question

What does this mean in PL?

In PL, it means showing that from a knowledge base KB, i.e., a (possibly extensive) formula in PL, a certain formula Q follows.

Entailment

We want to take existing knowledge from a knowledge base (KB) and obtain new knowledge or perform queries (Q) on it.

Definition

A formula KB *entails* a formula Q, or Q *follows from* KB, if every model of KB is also a model of Q. We write $KB \models Q$.

That is, in every interpretation in which KB is true, Q is also true.

Remark

Tautologies are always true, without restriction on the interpretation on the formula. If we denote by $\emptyset$ the empty formula, we have, for a tautology T, $\emptyset \models T$. For short, we write $\models T$.

Deduction theorem

This theorem allows us to link the semantic concept of entailment to the syntactic implication $\Rightarrow$.

Theorem

$A \models B$ if and only if $\models A \Rightarrow B$.

This theorem also provides a systematic way to verify whether $KB \models Q$: we can demonstrate that $KB \Rightarrow Q$. The latter is easy to verify and automatise: it suffices to build the truth table of the formula "$KB \Rightarrow Q$".

A big drawback of the truth table method

If n propositional variables are involved in a formula, the corresponding truth table will have 2^n rows. In the worst case, verifying if a formula is a tautology will require calculating every single row. The following table gives an idea of the time needed for a naive algorithm to operate on a formula, with the generous assumption that it can compute a row of the truth table in a millionth of a second.

propositional variables	rows	time required
1	$2^1 = 2$	2×10^{-3} ms
2	$2^2 = 4$	4×10^{-3} ms
10	$2^{10} = 1024$	1 ms
20	$2^{20} = 1048576$	1 s
30	$2^{30} = 1073741824$	18 minutes
40	$2^{40} = 1099511627776$	2 years
50	$2^{50} = 1125899906842624$	2142 years

Proof by contradiction

Theorem

$KB \models Q$ if and only if $KB \wedge \neg Q$ is unsatisfiable.

To show that the query Q follows from the knowledge base KB, we can add the negated query $\neg Q$ to the knowledge base and derive a contradiction, such as $A \wedge \neg A$. Since $A \wedge \neg A \equiv f$, a contradiction is unsatisfiable: therefore, by the above theorem, $KB \models Q$ has been proved. This proof procedure is frequently used in mathematics and also used in various automatic proof calculi such as the ones used in PROLOG.

Derivations and calculi

Instead of computing truth tables, we can instead manipulate formulas syntactically, and try to simplify KB by means of inference rules so that we can immediately conclude that $KB \models Q$.

Definition

This syntactic process is called *derivation*, and we write $KB \vdash Q$.

Definition

Such syntactic proof systems are called *calculi*.

They are similar to the syntactical derivations we find in elementary algebra:

$$(x + y)^2 = x^2 + 2xy + y^2.$$

Soundness and completeness

Definition

- A calculus is called *sound* if every derived proposition follows semantically, i.e., if $KB \vdash Q$, then $KB \models Q$.
- A calculus is called *complete* if all semantic consequences can be derived syntactically, i.e., if $KB \models Q$, then $KB \vdash Q$.

In other words, a sound calculus does not produce "false consequences". A complete calculus, on the other hand, ensures that the calculus does not overlook anything, i.e., if Q follows semantically from KB, then the calculus will find a syntactic proof for it.

Diagram summary

A good calculus should be both sound and complete. In such a case, syntactic derivation and semantic entailment are two equivalent relations. In the following diagram, $\mathrm{Mod}(A)$ designates the set of models for the formula A.

$$\begin{tikzcd}
KB \arrow[r, "\vdash", "\text{derivation}"'] \arrow[d, "\text{interpretation}"'] & Q \arrow[d, "\text{interpretation}"] & \begin{array}{c}\text{syntactic level}\\ \text{(formula)}\end{array} \\
\mathrm{Mod}(KB) \arrow[r, "\models", "\text{entailment}"'] & \mathrm{Mod}(Q) & \begin{array}{c}\text{semantic level}\\ \text{(interpretation)}\end{array}
\end{tikzcd}$$

Modus Ponens

We give an example of an important derivation rule for PL: the *modus ponens* (from Latin modus ponendo ponens, "mode where affirming affirms").

Definition

If A and $A \Rightarrow B$ are valid, we can derive B. We write it formally as:

$$\frac{A \qquad A \Rightarrow B}{B}\ \mathrm{MP}\,.$$

Example

$$\frac{rain \qquad rain \Rightarrow wet}{wet}\ \mathrm{MP}\,.$$

Resolution rule

We give an example of another important derivation rule for PL: the *resolution rule*. It can be seen as a generalisation of the modus ponens: we obtain it after setting $A = f$ and because $\neg B \vee C \equiv B \Rightarrow C$.

Definition

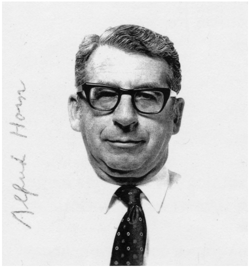
We write it formally as:

$$\frac{A \vee B \qquad \neg B \vee C}{A \vee C}\ \mathrm{Res}.$$

Example

$$\frac{\mathit{sun} \vee \mathit{rain} \qquad \neg \mathit{rain} \vee \mathit{wet}}{\mathit{sun} \vee \mathit{wet}}\ \mathrm{Res}.$$

Horn clauses: what and why


Alfred Horn circa 1973.

- Horn clauses are a special kind of formulas with big constraints on their form.
- These constraints make it easier (faster) for automatic proof systems to reason with and manipulate them (c.f. PROLOG).

Conjunctive normal form

There can be several ways to express the same thing using a formula. To simplify things for the purpose of, e.g., automatic proof systems, we constraint the representations.

Definition

A formula is in *conjunctive normal form* (CNF) if and only if it consists of a *conjunction of clauses*

$$K_1 \wedge K_2 \wedge \cdots \wedge K_m.$$

A *clause* K_i consists of a *disjunction of literals*

$$L_{i1} \vee L_{i2} \vee \cdots \vee L_{in_i}.$$

A *literal* is a variable or a negated variable.

"Completeness" of representations

Very nicely, the CNF does not restrict the set of formulas.

Theorem

Every PL formula can be transformed into an equivalent CNF.

Example

$$
\begin{aligned}
& A \vee B \Rightarrow C \wedge D \\
& \equiv \neg(A \vee B) \vee (C \wedge D) && \text{(implication)} \\
& \equiv (\neg A \wedge \neg B) \vee (C \wedge D) && \text{(de Morgan)} \\
& \equiv (\neg A \vee (C \wedge D)) \wedge (\neg B \vee (C \wedge D)) && \text{(distributivity)} \\
& \equiv ((\neg A \vee C) \wedge (\neg A \vee D)) \wedge ((\neg B \vee C) \wedge (\neg B \vee D)) && \text{(distributivity)} \\
& \equiv (\neg A \vee C) \wedge (\neg A \vee D) \wedge (\neg B \vee C) \wedge (\neg B \vee D) && \text{(associativity)}
\end{aligned}
$$

Horn clauses

Horn clauses are a special kind of formulas which are very useful in formal reasoning and proof systems.

Definition

Clauses with at most one positive literal, i.e., of the forms

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B) \quad \text{or} \quad (\neg A_1 \vee \cdots \vee \neg A_m) \quad \text{or} \quad B$$

or, equivalently, of the forms

$$A_1 \wedge \cdots \wedge A_m \Rightarrow B \quad \text{or} \quad A_1 \wedge \cdots \wedge A_m \Rightarrow f \quad \text{or} \quad B$$

are named *Horn clauses*, after their inventor. A clause with a single positive literal is a *fact*. In clauses with negative and one positive literal, the positive literal (here, B) is called the *head*.

Example

To motivate the use of Horn clauses in modelling and reasoning, consider the following knowledge base that describes the behaviour of a robot trying to grab a can of soda on a table:

$$
\begin{aligned}
KB = \{&oiled, && \text{(the robot is well-oiled)}\\
&can\_reach, && \text{(the robot is close enough to reach the can)}\\
&oiled \Rightarrow can\_move, && \text{(if the robot is well-oiled then it can move)}\\
&can\_reach \land can\_move \Rightarrow can\_grab\}.
\end{aligned}
$$

Note that KB can be seen as set of propositions; to recover a singular formula, we may simply use the conjunction of all the propositions in the set. We want to know whether the robot can grab the soda can, query which we may write as

$$can\_grab?$$

Example: derivation

To prove that *can_grab* holds, we can use the following generalised modus ponens rule:

$$\frac{A_1 \wedge \cdots \wedge A_m \qquad A_1 \wedge \cdots \wedge A_m \Rightarrow B}{B}\ \mathrm{MP}_m.$$

The derivation for *can_grab* can be written as follows. We first derive *can_move* from *oiled* and *oiled* $\Rightarrow$ *can_move* :

$$\frac{oiled \qquad oiled \Rightarrow can\_move}{can\_move}\ \mathrm{MP}.$$

The proposition *can_move* has thus been added to our knowledge base. Then, we can apply again the generalised modus ponens and the last proposition in the knowledge base:

$$\frac{can\_reach \wedge can\_move \qquad can\_reach \wedge can\_move \Rightarrow can\_grab}{can\_grab}\ \mathrm{MP}_m.$$

Drawback of modus ponens

Modus ponens is easy to apply and yields a complete (and sound) calculus for Horn clauses. However, when the knowledge base is large, it can derive many useless formulas if one begins with the wrong clauses. Instead of starting with facts and deriving to the query (*forward chaining*), we can instead start with the query and work backwards until the facts are reached (*backward chaining*).
SLD is a method to perform backward chaining on Horn clauses in particular.

SLD

Definition

A *definite clause* is a particular kind of Horn clause that contains a single positive literal, i.e., of the forms

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee Q) \quad \text{or} \quad Q.$$

Definition

SLD stands for *Selection rule driven Linear resolution for Definite clauses*.

To perform SLD, given a query Q?, we add a negated query such as $Q \Rightarrow f$ and try to derive a contradiction.
In PROLOG, programs consist in predicate logic Horn clauses, which are processed using SLD.
The search space is greatly reduced because less formulas are explored.

Example: derivation I

Let us go back to our robot trying to grab a can of soda. The knowledge base is still

$$\begin{aligned} KB = \{ & oiled, \\ & can\_reach, \\ & oiled \Rightarrow can\_move, \\ & can\_reach \wedge can\_move \Rightarrow can\_grab, \\ & can\_grab \Rightarrow f\}, \end{aligned}$$

augmented with the query $can\_grab \Rightarrow f$. The derivation can be written as follows.

$$\frac{can\_grab \Rightarrow f \qquad can\_reach \wedge can\_move \Rightarrow can\_grab}{can\_reach \wedge can\_move \Rightarrow f}.$$

Example: derivation II

Also,

$$\frac{can\_reach \wedge can\_move \Rightarrow f \qquad can\_reach}{can\_move \Rightarrow f},$$

and

$$\frac{can\_move \Rightarrow f \qquad oiled}{oiled \Rightarrow f},$$

but

$$\frac{oiled \Rightarrow f \qquad oiled}{()}$$

with () denoting the empty clause, i.e., a contradiction:

$$\begin{aligned}(oiled \Rightarrow f) \wedge oiled &\equiv (\neg oiled \vee f) \wedge oiled \\ &\equiv (\neg oiled \wedge oiled) \vee (f \wedge oiled) \\ &\equiv (\neg oiled \wedge oiled) \vee f \\ &\equiv (\neg oiled \wedge oiled).\end{aligned}$$

Computability and complexity

This section contains a (very!) short introduction to computability and complexity.
What you should remember from this section is summarized in slide 49.

Two big questions

In AI and robotics in particular, we are confronted to a lot of problems we, or the robot, might want an answer to, and fast.

Can I grab this can of coke?
Is it the can of coke that I needed?
What's the fastest way to get to the can of coke?
Am I alive?

Computability: is a problem decidable?

Is it even possible to get a solution?

(Computational) complexity

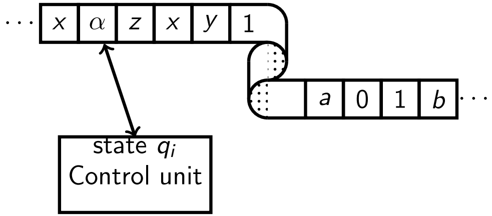
If we can get a solution, how difficult/costly is it to get it?

Turing machines: introduction

Turing machines (TM) are an abstract but quite simple model of computation, introduced by Alan Turing in 1936 in his paper *On Computable Numbers, with an Application to the Entscheidungsproblem*. They provide a *powerful* model of computation: we have yet to find a problem that cannot be tackled by a TM (provided the problem can be solved). They also provide a strong mathematical basis for many computer science concepts. Their simplicity allows researchers to precisely define, for instance, the *cost* of a certain computation/program. Just like for computers, the cost in time will be the number of steps a certain TM needs to perform a certain task, and the cost in space will be the amount of "memory" the machine requires.

Alan Turing circa 1938.

Turing Machines: informal description



A TM in its simplest form is comprised of a *workspace* (a tape, infinite to the right and to the left), a *head* (moves along the tape, and can read/modify it), and *internal states* q_i (in which the TM can be). The machine works as follows:

1. read what is written on the tape under the head,
2. change internal state accordingly,
3. perform an action according to this new state,
4. repeat.

Church-Turing thesis I

Alonzo Church.

Paraphrased:

> *Anything that can be computed by human means can be computed by a TM.*

- This includes doing a division by hand, performing an addition with an abacus, but also playing chess, calculators, and computers!

Church-Turing thesis II

- There exist other models of computations, such as circuits, rewriting systems, programming languages, λ-calculus, etc., but they are not more powerful than TMs: there are no problems that they can compute that cannot be computed by a TM.
- Note that the thesis is not a theorem: it identifies an informally and intuitively defined class of problems with a mathematically defined class.
- This identification is generous. No limit is placed on computation time, memory, etc., so if Church's thesis is right, and if we prove that something cannot be computed by a TM, allowing more memory or a longer computation time will not change the conclusion.
- The equivalences between different models of computation and the simplicity of TMs motivates the conception of computation being seen as simple symbol manipulation, be it in a computer, or in the (animal) brain. It makes it reasonable to abstract from differences between circuits, silicon chips, symbols, neurons..., despite their differences.

Uncomputability

Turing machines split the space of problems in two:

- problems they can solve (*computable* problems),
- problems they cannot (*uncomputable* problems).

Some uncomputable problems have a deceptively simple description, just as some arithmetic problems can be easily described but extremely hard to prove. They are mathematically well defined, but cannot be solved by a TM.

Halting problem

The following problem is uncomputable.

Example

Does a certain TM halt on a certain input?

Sometimes, it is easy to see whether a program (or a TM) will stop or loop forever on a certain input:

```
input = True
while(input):
    print "oops"
```

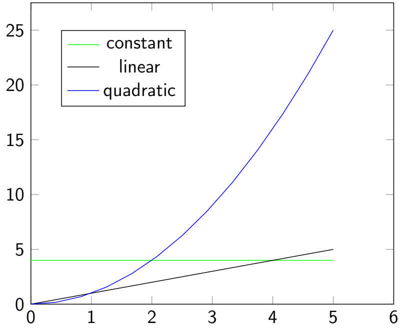
Such a Halting problem-solving machine would be tremendously useful, for engineers to check whether their airplane software will loop, for instance. Unfortunately, this problem is also uncomputable.

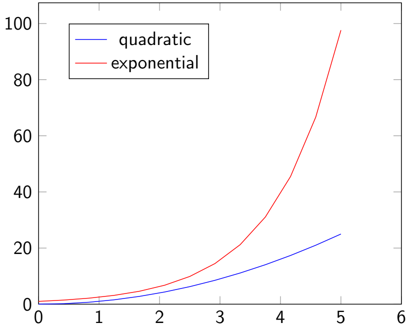
A little asymptotic complexity...

The consumption of memory or size of programs, machines, etc. can often be modelled by a function.

Question

How to compare the growth of two functions?





Polynomial time

In order to measure the complexity of a process, of a program, of an algorithm, etc., we can evaluate the time it takes to run. Thanks to TMs, all we need is to count the number of actions it makes until it stops after having written the result on its tape.

Definition

P is the set of problems that can be solved in polynomial time.

Example

- GCD (Greatest Common Divisor) is in P: Euclid's algorithm runs in quadratic time.
- PRIME ("Is a certain integer a prime number?") is in P.

NP

Definition

NP is the set of problems for which we can verify a solution in polynomial time.

Example

SAT, the problem that takes as input a formula in PL and returns t if the formula is satisfiable, is in NP (Cook-Levin 1973). Here, a solution is a model: to verify if the model indeed satisfies the formula takes a polynomial amount of time.

P versus NP

We know that $\mathrm{P} \subseteq \mathrm{NP}$, but knowing whether $\mathrm{P} = \mathrm{NP}$ is still an open problem. The standard opinion is that $\mathrm{P} \neq \mathrm{NP}$ and that it is a very complicated problem to solve, but opinions diverge on how to approach it.

What does it mean for us?

- The truth table method can determine every model of any formula in finite time: thus, the sets of *unsatisfiable, satisfiable, valid* formulas are computable.
- However, the computation time of the truth table grows *exponentially* in the number of variables of the formula.
- Are there better algorithms?
- Probably not (except if P=NP!): 3-SAT, the set of all (satisfiable) CNF formulas whose clauses have exactly three literals, is NP-*complete (Cook-Levin theorem)*: is amongst the most difficult problems of NP.

Next week...

- PL is employed in simple AI applications, but modelling real situations quickly requires a large number of propositional variables.
- We will be looking at *first-order logic* (FOL), which allows us to describe more complex and precise logical connections by adding onto the syntax and semantics of PL.
- We will do sensibly the same thing: define syntax, semantics, and how to perform inference in a FOL system.

RO47014 - Week 2: First-Order logic Course

Pierre Mercuriali
Responsible instructor: Carlos Hernandez Corbato

Spring 2022

From propositional logic to first-order logic

Consider the following sentence that describes a situation a robot is in.

"The robot 4 *is at position* $(32, 10)$*."*

In propositional logic, one could model this situation using the propositional variable

`robot_4_at_position_32_10`.

However, only 10 robots in a 100x100 grid would already require $10 \times 100 \times 100 = 10^5$ different variables to capture all the different positions!

Relations between objects are also costly to represent:

"The robot A is to the right of the robot B"

would require painstakingly entering, in our knowledge base, all the pairs of coordinates that verify the condition!

The power of first-order logic

- Universal knowledge and rules can be expressed very compactly thanks to *variables* and *functions*. In the preceding example, one can thus very easily access the position of robot #4 using a function symbol: $position(robot4)$. Thus there is no need to introduce a new symbol for every possible position.
- It is also possible to express relations between objects: $is\_to\_the\_right(robot1, robot2)$ and to define these relations:

$$\forall r1\ \forall r2\ is\_to\_the\_right(r1, r2) \Leftrightarrow \\ \exists x_{r1}\ \exists y_{r1}\ \exists x_{r2}\ \exists y_{r2} \\ position(r1, x_{r1}, y_{r1}) \wedge position(r2, x_{r2}, y_{r2}) \wedge x_{r1} > x_{r2},$$

which can be read as:

"For all r1 and r2, r1 is to the right of r2 if and only if there exist (x_{r1}, y_{r1}) and (x_{r2}, y_{r2}) such that r1 is at (x_{r1}, y_{r1}), r2 is at (x_{r2}, y_{r2}), and $x_{r1} > x_{r2}$."

First-order logic: overview

FOL: syntax

FOL: semantics

FOL in practice: Logical reasoning

Decidability

How to write a term?

In order to build FOL formulas, we need to formalise the notion of *term*, as it is one of the building blocks of our formulas.

Definition (Terms)

Let V be a set of variables, K a set of constants, F a set of function symbols. V, K, F are pairwise disjoints. The set of *terms* is defined recursively as follows:

- All variables and constants are (atomic) terms.
- If $t_1, \dots, t_n$ are terms and f is an n-place (*n-ary*) function symbol, then $f(t_1, \dots, t_n)$ is also a term.

Example

$f(\cos(\frac{\pi}{2}))$, $f(g(x))$, $battery\_state(\texttt{Robot1})$, are terms.

Predicate logic formulas

Definition

Let P be a set of predicate symbols. *Predicate logic formulas* are built as follows:

- If $t_1, \ldots, t_n$ are terms and p an n-place predicate, then $p(t_1, \ldots, t_n)$ is an (atomic) formula.
- If A, B are formulas, then $\neg A, (A), A \wedge B, A \vee B, A \Rightarrow B, A \Leftrightarrow B$ are formulas.
- If x is a variable and A is a formula, then $\forall x\ A$ and $\exists x\ A$ are formulas. $\forall$ and $\exists$ are known as the *universal* and *existential quantifiers*, respectively.
- $p(t_1, \ldots, t_n)$ and $\neg p(t_1, \ldots, t_n)$ are called literals.

FOL is extensively used in mathematics. It is a powerful and concise language to express properties and properly and unambiguously define objects, as in:

$$\forall n\ \mathrm{prime}(n) \Leftrightarrow (\forall m\ \mathrm{divides}(m, n) \Rightarrow (\mathrm{equals}(m, n) \vee \mathrm{equals}(m, 1))).$$

FOL formulas

Definition

Formulas in which every variable is in the scope of a quantifier are called *first-order sentences* or *closed formulas*. Variables which are not in the scope of a quantifier are called *free variables*.

Example (Scope)

The following formula is not closed:

$$p(x) \Rightarrow \forall y \ f(x, y).$$

Indeed, x is a free variable: it is not preceded by a quantifier or, in other words, it is not in the *scope* of a quantifier. However, y is a bound variable, quantified by $\forall y$.

Remark

CNFs and Horn clauses are defined similarly as they are in PL.

Examples

$likes(x, y)$ is to be read as: "x likes y".

Formula	Description
$\forall x\ frog(x) \Rightarrow green(x)$	All frogs are green
$\forall x\ (frog(x) \wedge brown(x)) \Rightarrow big(x)$	All brown frogs are big
$\exists x\ frog(x)$	There exists a frog
$\forall x\ likes(x, cake)$	Everyone likes cake
$\exists x\ likes(x, cake)$	Someone likes cake
$\neg(\forall x\ likes(x, cake))$	Not everyone likes cake
$\exists x\ \neg(likes(x, cake))$	Someone does not like cake
$\neg(\exists x\ likes(x, cake))$	There isn't someone who likes cake/ No one likes cake
$\forall x\ \neg likes(x, cake)$	Everyone dislikes cake

Examples

- Be aware that *likes* is not necessarily symmetric: $likes(x, y)$ is different from $likes(y, x)$!
- $older(x, y)$ is to be read as: "x is older than y".
- Note that *mother* is a function symbol here: $mother(Susan)$ refers to "Susans's mother".

Formula	Description
$\exists x\ \forall y\ likes(x, y)$	There is someone who likes everything
$\exists y\ \forall x\ likes(x, y)$	There is something that everyone likes
$\forall x\ robot(x) \Rightarrow likes(Susan, x)$	Susan likes every robot
$\exists x\ robot(x) \wedge likes(Susan, x)$	There is a robot whom Susan likes
$\exists x\ robotician(x) \wedge (\forall y\ robot(y) \Rightarrow likes(x, y))$	There is a robotician who likes every robot
$\forall x\ older(mother(x), x)$	Every mother is older than their child
$\forall x\ older(mother(mother(x)), x)$	Every grandmother is older than their child's child
$\forall x\ \forall y\ \forall z\ Rel(x, y) \wedge Rel(y, z) \Rightarrow Rel(x, z)$	*Rel* is a transitive relation

Notation

- It can be particularly useful to indicate the number of places (the arity) of a function or predicate symbol explicitly.
- In Prolog in particular, which we will be using, the arity is sometimes written right after the predicate:

$$\textit{stop}/1,$$
$$\textit{older}/2,$$
$$\textit{remove}/2.$$

Example

In this example, the predicate *looking* is used both in its intransitive form (as in, "I am looking") and in its transitive form (as in, "I am looking *at* something"). The difference lies the arity of the predicate.

Symbol(s)	Formula	Description
$\textit{looking}/1$	$\textit{looking}(\textit{robot}1)$	The robot 1 is looking.
$\textit{can}/1, \textit{looking}/2$	$\exists x\ \textit{can}(x) \wedge \textit{looking}(\textit{robot}1, x)$	The robot 1 is looking at a can.

Interpretations

- Recall that in PL, every variable is assigned a truth value by an interpretation.
- In FOL, we also interpret, starting with the building blocks of the FOL formulas: we map constants and variables to a set of names of objects in the world. Function symbols and predicate symbols are mapped to the set of functions and predicates in the world, respectively. The meaning of the formula is then recursively defined over the construction of the formula.

Definition

An *interpretation* $\mathbb{I}$ of a formula F is defined as a couple $(\Delta_{\mathbb{I}}, \cdot^{\mathbb{I}})$, where $\Delta_{\mathbb{I}}$ is a non-empty set, called the *domain of interpretation*, and $\cdot^{\mathbb{I}}$ is an *interpretation function*, that maps:

- a constant c in F to a constant $c^{\mathbb{I}} \in \Delta_{\mathbb{I}}$;
- a n-place function symbol ϕ in F to a n-ary function $\phi^{\mathbb{I}} : \Delta_{\mathbb{I}}^n \to \Delta_{\mathbb{I}}$;
- a n-place predicate symbol p in F to a n-ary relation $p^{\mathbb{I}}$ on $\Delta_{\mathbb{I}}$, that is, p is a subset of $\Delta_{\mathbb{I}}^n$, that we can consider as a function $p^{\mathbb{I}} : \Delta_{\mathbb{I}}^n \mapsto \{f, t\}$.

Interpretations: more on relations

"$(x_1, \ldots, x_n)$ are in relation through $p^{\mathbb{I}}$" is denoted either by

$$(x_1, \ldots, x_n) \in p^{\mathbb{I}} \quad \text{or by} \quad p^{\mathbb{I}}(x_1, \ldots, x_n).$$

Example

Consider the binary (2-place) relation *likes*, interpreted over the domain $\Delta_{\mathbb{I}} = \{Milton, Joe, Charity\}$. Then, $likes^{\mathbb{I}}$ is a subset of $\Delta_{\mathbb{I}}^2$, i.e., a subset of $\{(Milton, Milton), (Milton, Joe), (Milton, Charity), (Joe, Milton), (Joe, Joe), (Joe, Charity), (Charity, Milton), (Charity, Joe), (Charity, Charity)\}$; for instance, we may decide that $likes^{\mathbb{I}} = \{(Milton, Charity), (Milton, Joe), (Joe, Charity)\}$ to express the fact that Milton likes Charity, that Milton likes Joe, and that Joe likes Charity.

(For more information on Milton, Joe, and Charity, read Asimov's short story: "True Love").

Interpretation of a quantifierless formula

Example

Let c_1, c_2, c_3 be constants, *plus* a 2-place function symbol, and *greater* a 2-place predicate symbol. We will figure out the *truth* of the formula

$$F \equiv greater(plus(c_1, c_3), c2),$$

given two different interpretations $\mathbb{I}_1$ and $\mathbb{I}_2$.
First, consider the interpretation $(\Delta_{\mathbb{I}_1}, \cdot^{\mathbb{I}_1})$ where $\Delta_{\mathbb{I}_1} = \{1, 2, 3, 4, 5\}$ and where $\cdot^{\mathbb{I}_1}$ verifies:

$$\mathbb{I}_1 : c_1 \mapsto 1,\ \ c_2 \mapsto 2,\ \ c_3 \mapsto 3,\ \ plus \mapsto +,\ \ greater \mapsto > .$$

Once interpreted, the formula is mapped to $1 + 3 > 2$ or, once evaluated, $4 > 2$. $>$ is a binary relation and, over $\Delta_{\mathbb{I}_1} = \{1, 2, 3, 4, 5\}$, $>$ is the set $\{(2, 1), (3, 1), (3, 2), (4, 1), (4, 2), (4, 3), (5, 1), (5, 2), (5, 3), (5, 4)\}$.
Thus, since $(4, 2) \in >$, F is true under the interpretation $\mathbb{I}_1$.

Interpretation of a quantifierless formula

Example

Second, under the following interpretation $\mathbb{I}_2$, the formula

$$F \equiv greater(plus(c_1, c_3), c2),$$

is false: here, $\Delta_{\mathbb{I}_1} = \{1, 2, 3, 4, 5\}$ also, but $\cdot^{\mathbb{I}_2}$ verifies:

$$\mathbb{I}_2 : c_1 \mapsto 2,\ c_2 \mapsto 3,\ c_3 \mapsto 1,\ plus \mapsto -,\ greater \mapsto > .$$

Once interpreted, the formula is mapped to $2 - 1 > 3$ or, once evaluated, $1 > 3$. Since $(1, 3) \notin >$, F is false under the interpretation $\mathbb{I}_2$.

Truth of a formula

Let us formalise what we have done in the previous examples.

Definition (Truth value of a predicate)

- An atomic formula $p(t_1, \ldots, t_n)$ is *true* (or valid) under the interpretation $\mathbb{I}$ if, after interpretation and evaluation of all terms $t_1, \ldots, t_n$ and interpretation of the predicate p through the n-place relation $p^{\mathbb{I}}$, it holds that

$$(t_1^{\mathbb{I}}, \ldots, t_n^{\mathbb{I}}) \in p^{\mathbb{I}}.$$

- The truth of quantifierless formulas follows from the truth of atomic formulas, as in propositional calculus, through the semantics of the logical operators $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ defined in week 1.

Remark

You might also find the functional notation $\mathbb{I}(t)$ for $t^{\mathbb{I}}$.

Quantifiers

Definition

- A formula $\forall x\ F$ is true under the interpretation $\mathbb{I}$ exactly when it is true given an arbitrary change of the interpretation for the variable x, and only for x;
- a formula $\exists x\ F$ is true under the interpretation $\mathbb{I}$ exactly when there is an interpretation for x that makes the formula true.

In other words,

- $\forall x\ F$ is interpreted as "for all $x \in \Delta_{\mathbb{I}}, F$", and
- $\exists x\ F$ is interpreted as "there exists $x \in \Delta_{\mathbb{I}}$ such that F".

Interpretation of a quantified formula

Example

Consider the following formula:

$$\forall x \; \forall y \;\; p(a,x) \wedge p(x,y) \Rightarrow p(a,y).$$

1. Under the interpretation $\Delta_{\mathbb{I}}$ = every human being, $a^{\mathbb{I}}$ = me, and $p^{\mathbb{I}}(x,y)$ = "y is a friend of x", the formula can be understood as
   "The friends of my friends are my friends".
2. Under the interpretation $\Delta_{\mathbb{I}} = \mathbb{N}$, $a^{\mathbb{I}} = 5$, and $p^{\mathbb{I}}(x,y) = (x \leq y)$, the formula is interpreted as
   "For all x and for all y, if $x \geq 5$ and $y \geq x$, then $y \geq 5$"

   which it true, because of the transitivity of $\geq$.

Satisfiability

The definitions of semantic equivalence of formulas, satisfiability, validity, unsatisfiability, model, and semantic entailment, are the same as in PL. For instance,

Definition

Let F be a formula in FOL and $\mathbb{I}$ an interpretation of F. $\mathbb{I}$ is a *model* of F, which we denote by $\mathbb{I} \models F$, if F is true in this interpretation.

Example

With the formula from the previous example, consider the interpretation $\Delta_{\mathbb{I}} = \mathbb{N}$, $a^{\mathbb{I}} = 5$, and $p^{\mathbb{I}}(x, y) = (x = y + 1)$, the formula is interpreted as

"For all x and for all y, if $5 = x + 1$ and $x = y + 1$, then $5 = y + 1$".

Then, $\mathbb{I} \not\models F$, because of the counter-example $5 = 4 + 1$ and $4 = 3 + 1$, but $5 \neq 3 + 1$.

Useful equivalences

Given a formula of FOL F (either closed or open),

$$\neg\forall x\ \neg F \equiv \exists x\ F \tag{1}$$
$$\neg\forall x\ F \equiv \exists x\ \neg F \tag{2}$$
$$\neg\exists x\ F \equiv \forall x\ \neg F \tag{3}$$
$$\forall x\ \forall y\ \equiv \forall y\ \forall x \tag{4}$$
$$\exists x\ \exists y\ \equiv \exists y\ \exists x \tag{5}$$

Example

Formula	Description
$\neg(\forall x\ likes(x, cake))$	Not everyone likes cake
$\exists x\ \neg(likes(x, cake))$	Someone does not like cake
$\neg(\exists x\ likes(x, cake))$	There isn't someone who likes cake
$\forall x\ \neg likes(x, cake)$	Everyone dislikes cake

Be careful with the order of alternating quantifiers!

Compare the following two formulas:

- $\forall x \, \exists y \; needs(x, y)$, that can be understood as
  "Everybody needs somebody";
- $\exists x \, \forall y \; needs(x, y)$, that can be understood as
  "There is someone who needs everyone".

When dealing with alternating quantifiers ($\exists x \, \forall y \, \exists z \; \ldots$ the order in which they are understood is very important, order which we can make clearer using parentheses:

- $\forall x \; (\exists y \; needs(x, y))$,
- $\exists x \; (\forall y \; needs(x, y))$.

(For more information regarding the first sentence, refer to Everybody Needs Somebody by the Blues Brothers).

Equality

- Recall that we defined atomic formulas as $p(t_1, \ldots, t_n)$ with p an n-place predicate, and $t_1, \ldots, t_n$ terms.
- In FOL, we can also use the *equality symbol* to indicate that two terms refer to the same object. It can be seen as a predicate *equals*/2, and written with the simple equality symbol: $=$.
- The predicate *equals*/2 is the set of all pairs of objects in which both elements of the pair are the same object.

Example

The formula

$$inventor(lambda\_calculus) = Alonzo\_Church$$

expresses that the inventor of λ-calculus and Alonzo Church are the same.

Logical reasoning in FOL

- In PL, we have seen some ways to reason with formulas using proof calculi: we can derive new knowledge from a knowledge base using inference rules such as the Modus Ponens and the Resolution rule.
- With the introduction of variables and quantification, it becomes much more complex to reason like that, and we need new ways to manipulate formulas, such as rules to eliminate quantifiers.

Question

What are some concrete issues that automatic theorem provers, that is, implementation of proof calculi, have to tackle to reason with knowledge bases?

Question

How do automatic theorem provers deal with these issues?

Variable substitution

Consider the following knowledge base

$$
\begin{aligned}
KB = \{\forall x \; type(x, Bowl) \Rightarrow type(x, Container), \qquad (6)\\
type(Bowl01, Bowl)\} \qquad (7)
\end{aligned}
$$

that contains a rule that states that every object of type bowl is also an object of type container, as well as a fact that assigns the entity *Bowl*01 to the type *Bowl*.

Question

How to prove that

$$type(Bowl01, Container)$$

also holds?

(Perhaps straightforward for us, but not so much for a machine!)

Variable substitution: Herbrand universe

Definition (Substitution)

Let x be a (free) variable, X a symbol, and Φ a sentence in FOL. Assigning symbol X to the variable x in the sentence Φ is called a *substitution*, and is written as $\Phi[x/X]$.

The reasoning problem in FOL can be phrased as a search problem in the space of possible variable substitutions, which is called the *Herbrand universe*. Here, consider the substitution $[x/Bowl01]$ in sentence (6) in the preceding slide.

$$\begin{aligned} &(\ type(x, Bowl) \Rightarrow type(x, Container))[x/Bowl01] \\ \equiv\ & type(Bowl01, Bowl) \Rightarrow type(Bowl01, Container) \\ \equiv & \neg\ type(Bowl01, Bowl) \vee type(Bowl01, Container), \end{aligned}$$

which is satisfied if and only if $type(Bowl01, Container)$ holds.

Propositionalization

Eliminating variables by substitution through constant or function symbols is called *propositionalization*: a sentence whose variables have been substituted corresponds to an expression in PL.

Resolution

- Resolution calculus is historically the first automatized calculus, as well as the most important efficient calculus for formulas in CNF.
- It works much the same as in the propositional case: one adds a negated query to the knowledge base, and tries to reach a contradiction.
- It is motivated by results from the 70s on correctness and completeness which makes it a natural candidate for reasoning tasks in AI, together with the power of predicate logic.

Example of resolution

Consider the following knowledge base, that expresses that everyone knows their own mother.

$$KB = \forall x \; knows(x, mother(x)).$$

We now ask whether Henry knows anyone, i.e., whether

$$\exists y \; knows(henry, y).$$

We thus want to derive a contradiction from

$$knows(x, mother(x)) \wedge \neg knows(henry, y).$$

This is obtained by the following substitutions:
$\{[x/henry], [y/mother(henry)]\}$, as we obtain the following contradiction:

$$knows(henry, mother(henry)) \wedge \neg knows(henry, mother(henry))$$

from which we can derive the empty clause using a resolution step.

Unification

Definition (Unification)

Two literals are called *unifiable* if there is a substitution σ for all variables which makes the literals equal, called the *unifier*. A unifier is called the *most general unifier* (MGU) if all other unifiers can be obtained from it by substitution of variables.

The substitution step in the previous slide was a unification:

$$\begin{tikzcd}
knows(x, mother(x)) \arrow[dr] & & knows(henry, y) \arrow[dl] \\
& \boxed{\{[x/henry], [y/mother(henry)]\}} \arrow[d] & \\
& knows(henry, mother(henry)) &
\end{tikzcd}$$

Examples of unification

In this slide, a, b refer to constant symbols, x, y to variable symbols, and f, g to function symbols.

Expression	Unifying substitution	Explanation
$a = a$	any	all (tautology)
$a = b$	none	a and b do not match
$x = x$	any	all (tautology)
$a = x$	$[x/a]$	x is unified with the constant a
$f(a) = g(a)$	none	f and g do not match
$f(x) = f(y)$	$[x/y]$	x is unified with y
$f(g(x)) = f(y)$	$[y/g(x)]$	unifies y with the term $g(x)$
$x = y, y = a$	$[x/a], [y/a]$	unifies x and y with a

There exist several unification algorithms, the first of which was discovered by J. Herbrand.

Motivation for resolution

- With only a resolution rule and a factorization, both of which work by unification, the empty clause can be derived from any unsatisfiable formula in CNF.
- Therefore, if they can, queries can be answered by a theorem prover (and we get the proof!)
- However, the immense combinatorial search space makes it a costly problem.
- Some strategies attempt to tackle the problem of reducing the search space, such as applying rules to small literals first, or forcing a clause from $KB \wedge \neg Q$ to appear in all steps of the derivation (at the cost of completeness).

Decidability of PL

Theorem

All true entailments can be found. All false entailments can be refuted. (Truth table method)

Question

But what about FOL? We cannot use the truth table method!

Semi-decidability: towards description logics

- FOL is *semi-decidable* (Turing, Church, 1936), i.e., for every FOL formula F, the test $\models F$ (or $\vdash F$) is semi-decidable.
- In other words, there exists an algorithm that takes a formula in FOL F as input, and behaves thusly:
  - If $\vdash F$, then the algorithm returns *true*;
  - Otherwise, either the algorithm returns *false*, or it never stops.
- There is no algorithm that will systematically decide whether $\models F$ in finite time: i.e., this test is undecidable.
- Usually, a robot only takes a decision after its reasoning tasks have been completed. It's better to do the reasoning in finite time!

For this reason, we define formalisms that are equivalent to *decidable fragments of FOL*, namely, Prolog, and description logics.

RO47014 - Week 3: Description logics Course

Pierre Mercuriali
Responsible instructor: Carlos Hernandez Corbato

Spring 2022

Description logics

We have seen two ways of modelling the real world.

- PL, which powerful but quickly gets unsuitable for robotics tasks;
- FOL, which is much more powerful and descriptive but quickly gets too complicated for reasoning.

Can we get the best of both worlds?

Description logics (DLs) are knowledge representation languages, developed since the 80s, and used a lot in knowledge representation, such as bioinformatics, where it assists in the handling of biomedical knowledge.

Description logics: overview

Propositional logic: syntax

In PL, our building blocks are *propositional variables*, that we combine using *logical operators*.

- Set of symbols Σ (propositional variables)
- Logical connectives: $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$
- Two special symbols: $\{t, f\}$
- $Op, \Sigma, \{t, f\}$ are pairwise disjoint.

Definition (Set of propositional formulas)

- t and f are formulas;
- all propositional variables are formulas;
- if A and B are formulas, then $\neg A$, (A), $A \wedge B$, $A \vee B$, $A \Rightarrow B$, and $A \Leftrightarrow B$ are formulas.

Propositional logic: semantics

To give meaning to formula, we *interpret* them. Is a certain formula true or false?

- An *interpretation* is a mapping $\Sigma \to \{t, f\}$.
- The truth value of a formula is computed recursively using the semantic definition of the operators, given by a truth table:

A	B	(A)	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
t	t	t	f	t	t	t	t
t	f	t	f	f	t	f	f
f	t	f	t	f	t	t	f
f	f	f	t	f	f	t	t

Definition

A formula is called

- *satisfiable* if it is true for at least one interpretation, then called a *model*;
- *valid* if it is true for all interpretations;
- *unsatisfiable* if it is not true for any interpretation.

Propositional logic: reasoning

- Computation rules: Modus Ponens, Resolution rule.
- There are sound and complete calculi!

In the figure: $\mathrm{Mod}(A)$ is the set of models for the formula A.

$$\begin{tikzcd}
KB \arrow[r, "\vdash", "\text{derivation}"'] \arrow[d, "\text{interpretation}"'] & Q \arrow[d, "\text{interpretation}"] \\
\mathrm{Mod}(KB) \arrow[r, "\models", "\text{entailment}"'] & \mathrm{Mod}(Q)
\end{tikzcd}$$

syntactic level
(formula)

semantic level
(interpretation)

Decidability: PL vs FOL

- In PL: *decidability*

  All true entailments can be found, all false entailments can be refuted (truth table method).

- In FOL: *semi-decidability*

  There is an algorithm that finds true entailments, but might not stop searching for a refutation for a false entailment.

First-order logic: syntax

In first-order logic, we have more building blocks than in propositional logic:

- Operators: same as in PL
- Quantifiers: universal ($\forall x$), existential ($\exists x$)
- Variables: x, y, z, etc.
- Constants: $Container, Bowl$
- Function symbols (access properties): $location, x\_coordinate$
- Predicates (return true or false): $robot, is\_to\_the\_left\_of, drone$

Example

"Every robot knows someone who likes it":

$$\forall x \; \exists y \; (robot(x) \Rightarrow (knows(x, y) \wedge likes(y, x)))$$

First-order logic: syntax (terms and formulas)

Definition (Terms)

- Variables and constants are terms;
- $f(t_1, \ldots, t_n)$ is a term, with f a function symbol, t_i terms.

Definition (FOL (predicate) formulas)

- $p(t_1, \ldots, t_n)$ is a formula, with p a predicate symbol, t_i terms;
- $\forall x\ A$ and $\exists x\ A$ are formulas, with A a formula, x a variable;
- $\neg A$, (A), $A \wedge B$, $A \vee B$, $A \Rightarrow B$, and $A \Leftrightarrow B$ are formulas, with A, B formulas (just like in PL).

First-order logic: semantics

- An *interpretation* $\mathbb{I}$ is a couple $(\Delta_{\mathbb{I}}, \cdot^{\mathbb{I}})$;
- $\Delta_{\mathbb{I}}$ is the *domain of interpretation* (integers, real numbers, sets of people, all the objects in the world, etc.), $\cdot^{\mathbb{I}}$ the *interpretation function*:
  - a constant symbol c is interpreted as a constant $c^{\mathbb{I}}$ in $\Delta_{\mathbb{I}}$;
  - a n-place function symbol f is interpreted as a n-ary function $f^{\mathbb{I}} : \Delta_{\mathbb{I}}^{n} \to \Delta_{\mathbb{I}}$;
  - a n-place predicate symbol p is interpreted as a n-ary relation, and can be considered as a function $p^{\mathbb{I}} : \Delta_{\mathbb{I}}^{n} \to \{t, f\}$.
- A formula $p(t_1, \ldots, t_n)$ is *true under the interpretation* $\mathbb{I}$ if it holds that $p^{\mathbb{I}}(t_1^{\mathbb{I}}, \ldots, t_n^{\mathbb{I}}) = t$.

First-order logic: interpretations

Example

Consider the formula $F = \forall x\ \forall y\ p(x,y) \Rightarrow p(y,x)$.

1. - $\Delta_{\mathbb{I}_1} = \mathbb{N}$
   - $p^{\mathbb{I}_1} ==$ (the "equality" over natural numbers)

   The formula is understood as

   $$\forall x, y \in \mathbb{N}, x = y \Rightarrow y = x.$$

   Then, $\mathbb{I}_1 \models F$.
2. - $\Delta_{\mathbb{I}_2} =$ the set of all humans in Shakespeare's plays
   - $p^{\mathbb{I}_2} = loves/2$

   The formula is understood as
   "if an individual loves another, then the latter also loves the former".

   Then, $\mathbb{I}_2 \not\models F$ (counter-example: Helena loves Bertram in *All's Well That Ends Well*, but Bertram does not love her back).

Why description logics? Towards $\mathcal{ALC}$

Pro(s) of FOL

- High expressivity

Cons of FOL

- Proofs are very complex (semi-decidability)
- It can be difficult to find a consensus for representations (many ways to describe the same thing)

Between PL and FOL

One solution is a compromise between expressivity and scalability:

fragments of FOL.

- More expressive than PL,
- Yet simple enough to allow decidable reasoning!

First, we will consider a simple DL called $\mathcal{ALC}$.
Then, we will progressively add more expressiveness to reach OWL.

Description logic vocabulary

Question

What are our building blocks? What do we manipulate?

Here, it is convenient to consider right now a certain interpretation, $\mathbb{I}$.

- *Concepts*: a concept $\mathtt{C}$ is interpreted as a set $\mathtt{C}^{\mathbb{I}}$ of things we want to model (the set of all natural numbers, a set of robots, a set of soup cans... c.f. FOL);
- *Roles*: a role $\mathtt{r}$ is interpreted as a *binary* relation $\mathtt{r}^{\mathbb{I}}$ (notice here that we are only concerned with binary relations; compare with FOL!);
- *Instances*: an instance $\mathtt{a}$ is interpreted as an individual $\mathtt{a}^{\mathbb{I}}$ in the real world.

Example of DL concepts

Example

- the concept of `Robot`, interpreted as the set of all robots;
- the role `hasComponent`, used to indicate that a certain robot has a certain other component, thereby linking instances together;
- the instance `spot1` for the specific SpotMini robot that was used in Boston Dynamic's 2018 youtube video, a photo of which is given here.

Figure 1: Source : Wikipedia

Set theory

Once interpreted, concepts correspond to sets; naturally, naive set theory connectives such as $\cap, \cup$, etc. are available in $\mathcal{ALC}$ with the connectives $\sqcap, \sqcup$, etc.:

$$(\mathtt{C} \sqcup \mathtt{D})^{\mathbb{I}} = \mathtt{C}^{\mathbb{I}} \cup \mathtt{D}^{\mathbb{I}}.$$

Example

The concept of blue robots, at the "intersection" of the concept of blue things and the concept of robots:

$$\mathtt{Blue} \sqcap \mathtt{Robot}.$$

Knowledge base in $\mathcal{ALC}$

Here is an example of a knowledge base in $\mathcal{ALC}$. The symbols will be properly defined in the following slides.

Tbox ("terminology box")

Terminological knowledge: knowledge about the concepts of a domain

$\texttt{Robot} \sqsubseteq \texttt{ArtificialConstruct}$ (a robot is an artificial construct)

$\texttt{Skillet} \sqsubseteq \texttt{CookingVessel}$ (a skillet is a cooking vessel)

$\texttt{Hagelslag} \equiv \texttt{SweetFood} \sqcap \exists \texttt{ingredient.Chocolate} \sqcap \texttt{Small}$
(hagelslag is *exactly* a sweet, small food made of chocolate)

ABox ("assertion box")

Assertional knowledge: knowledge about individuals or entities

`Hagelslag(hagelslag1)` (whatever is referred to by hagelslag1 is a hagelslag)

`hasIngredient(chocolate1,hagelslag1)` (what is referred to by hagelslag1 contains what is being referred to by chocolate1)

Syntax of $\mathcal{ALC}$

$\mathcal{ALC}$

*A*ttributive (Concept) *L*anguage with *C*omplements

Like in PL and FOL, we start with the building blocks (concepts/classes, roles, and instances) and combine them to make more complex formulas.

We also have two special concepts:

- $\top$, read as "top": the *universal concept*, the most general concept;
- $\bot$, read as "bottom": the *bottom concept*, the unsatisfiable concept, also called the most specific concept.

Syntax (and intuitive semantics) of $\mathcal{ALC}$

The $\mathcal{ALC}$ concepts are syntactically defined as follow, with `C`, `D` two concepts, and `r` a role.

- $\top$ and $\bot$ are concepts.
- An *atomic concept*, that is, a concept name, is a concept.
- $\mathtt{C} \sqcap \mathtt{D}$ is the *conjunction* of the two concepts `C` and `D`, and represents the set of individuals that are in `C`, AND in `D`.
- $\mathtt{C} \sqcup \mathtt{D}$ is the *disjunction* of the two concepts `C` and `D`, and represents the set of individuals that are in `C`, OR in `D`.
- $\neg\mathtt{C}$ is called the *complementary* of `C` and represents the set of individuals that are NOT in `C`.
- $\exists\mathtt{r}.\mathtt{C}$ corresponds to the *existential quantification*: the set of individuals that are linked to at least one individual of `C` by `r`.
- $\forall\mathtt{r}.\mathtt{C}$ corresponds to the *universal quantification*: the set of individuals whose images by `r`, if any, are individuals of `C`. (The image of an individual by a role is all individuals linked to it by the role).

Examples: intersection and union

Example

- The concept of all graspable objects that can be discarded:

$$\texttt{Graspable} \sqcap \texttt{Junk}$$

- The concept of all food that are either fruits or vegetables:

$$\texttt{Fruit} \sqcup \texttt{Vegetable}$$

- The concept of things that are not fruit:

$$\neg\texttt{Fruit}$$

- The concept of food that is not fruit:

$$\texttt{Food} \sqcap \neg\texttt{Fruit}$$

Examples: (role) quantification

Example

- The concept of all things that contain chocolate:

$$\exists\texttt{hasIngredient}.\texttt{Chocolate}$$

- The concept of all food that contain electricity:

$$\texttt{Food} \sqcap \exists\texttt{contains}.\texttt{Electricity}$$

- The concept of robots that only contain chocolate:

$$\texttt{Robot} \sqcap \forall\texttt{contains}.\texttt{Chocolate}$$

$\exists\texttt{r}.\texttt{C}$ and $\forall\texttt{r}.\texttt{C}$ are sometimes called *restrictions* (existential and universal, respectively).

Knowledge base in $\mathcal{ALC}$

An $\mathcal{ALC}$ knowledge base is a set of $\mathcal{ALC}$ formulas in four possible forms.

Definition ($\mathcal{ALC}$ terminological axioms)

These axioms are given in a TBox.

- `C` $\sqsubseteq$ `D`, read as "the concept `C` *is subsumed by* the concept `D`", indicates that the concept `C` is more specific than the concept `D`;
- `C` $\equiv$ `D`, read as "the concepts `C` and `D` are *equivalent*". This axiom is particularly useful to define concepts, like the "equality" in FOL.

Definition ($\mathcal{ALC}$ assertional axioms)

These axioms are given in an Abox.

- `C(a)` means that `a` is an *instance* of `C`;
- `r(a, b)` means that `a` is *related* to `b` by the role `r`.

Examples of terminological axioms and FOL equivalent axioms

Example (TBox)

- Every robot is artificial:

$$\texttt{Robot} \sqsubseteq \texttt{Artificial}$$

$(\forall x\ robot(x) \Rightarrow artificial(x))$

- A robot is a mechanical device that is also a perceptual agent:

$$\texttt{Robot} \equiv \texttt{MechanicalDevice} \sqcap \texttt{PerceptualAgent}$$

$(\forall x\ robot(x) \Leftrightarrow mechanical\_device(x) \wedge perceptual\_agent(x))$

- Hagelslag is made with chocolate:

$$\texttt{Hagelslag} \sqsubseteq \exists \texttt{hasIngredient}.\texttt{Chocolate}$$

$(\forall x\ \exists y\ hagelslag(x) \Rightarrow (chocolate(y) \wedge has\_ingredient(x, y)))$

Examples of assertional axioms

Example (ABox)

- The instance tiago1 is a robot.

$$\texttt{Robot(tiago1)}$$

- The instance tiago1 is a blue mechanical device.

$$\texttt{(Blue} \sqcap \texttt{MechanicalDevice)(tiago1)}$$

- The instance tiago1 has at least one gripper amongst its components.

$$(\exists \texttt{hasComponent.Gripper)(tiago1)}$$

Model-theoretic semantics for $\mathcal{ALC}$

Time to give meaning to our formulas.

- We have already given some intuition on semantics using set theory: once interpreted, concepts correspond to sets.
- The semantics we are using here is based on the theory of models (there exist several others, but this is out of the scope of this course).
- In this section, we give a proper definition of *interpretations*, just like for PL and FOL.

Interpretations

Definition

An *interpretation* $\mathbb{I}$ is a pair $(\Delta_{\mathbb{I}}, \cdot^{\mathbb{I}})$ where

- $\Delta_{\mathbb{I}}$ is the domain of interpretation, i.e., a set of individuals,
- $\cdot^{\mathbb{I}}$ is an interpretation that maps
  - individual names $\mathtt{a}$ to domain elements $\mathtt{a}^{\mathbb{I}} \in \Delta_{\mathbb{I}}$;
  - concept names $\mathtt{C}$ to a subset of the domain elements $\mathtt{C}^{\mathbb{I}} \subseteq \Delta_{\mathbb{I}}$;
  - role names $\mathtt{r}$ to a set of pairs of domain elements (i.e., to a binary relation) $\mathtt{r}^{\mathbb{I}} \subseteq \Delta_{\mathbb{I}} \times \Delta_{\mathbb{I}}$.

Diagram: domain of interpretation

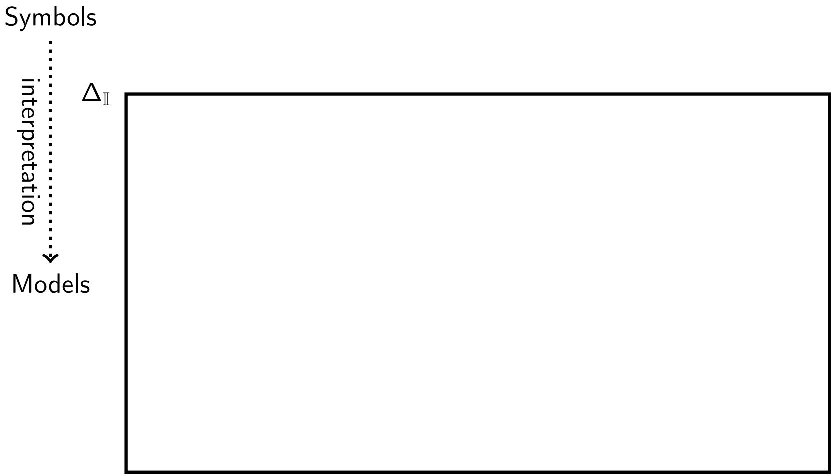


Diagram: concepts

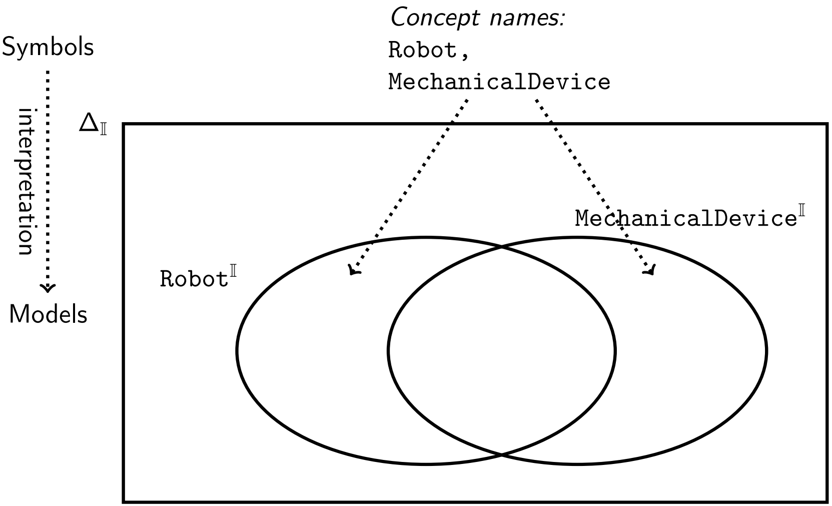


Diagram: concepts, instances

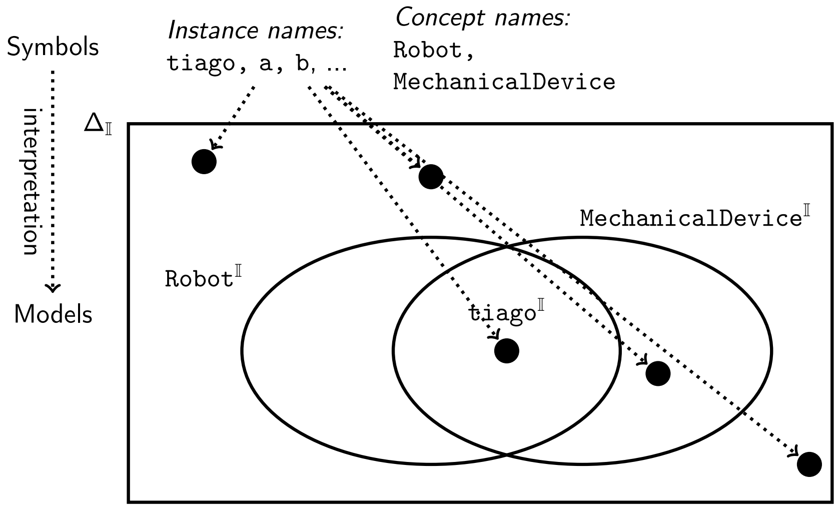
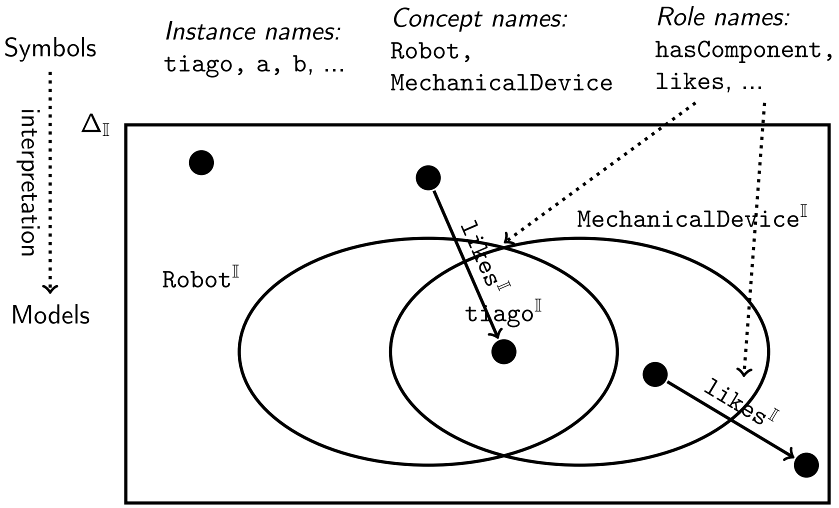


Diagram: concepts, instances, roles


Symbols
interpretation
Models
Instance names:
tiago, a, b, ...
Concept names:
Robot,
MechanicalDevice
Role names:
hasComponent,
likes, ...
$\Delta_{\mathbb{I}}$
$\texttt{MechanicalDevice}^{\mathbb{I}}$
$\texttt{Robot}^{\mathbb{I}}$
$\texttt{likes}_{\mathbb{I}}$
$\texttt{tiago}^{\mathbb{I}}$
$\texttt{likes}^{\mathbb{I}}$

Axioms for concept constructs

The interpretation function $\cdot^{\mathbb{I}}$ satisfies the following equalities. The set-theoretical notation $A \backslash B$, read "A minus B", is all the elements that are in A but NOT in B: $A \backslash B = \{x \in A | x \notin B\}$.

$$\top^{\mathbb{I}} = \Delta_{\mathbb{I}} \tag{1}$$
$$\bot^{\mathbb{I}} = \emptyset \tag{2}$$
$$(\mathtt{C} \sqcup \mathtt{D})^{\mathbb{I}} = \mathtt{C}^{\mathbb{I}} \cup \mathtt{D}^{\mathbb{I}} \tag{3}$$
$$(\mathtt{C} \sqcap \mathtt{D})^{\mathbb{I}} = \mathtt{C}^{\mathbb{I}} \cap \mathtt{D}^{\mathbb{I}} \tag{4}$$
$$(\neg \mathtt{C})^{\mathbb{I}} = \Delta_{\mathbb{I}} \backslash \mathtt{C}^{\mathbb{I}} \tag{5}$$
$$(\exists \mathtt{r}.\mathtt{C})^{\mathbb{I}} = \{x \in \Delta_{\mathbb{I}} \mid \exists y \in \mathtt{C}^{\mathbb{I}}, (x, y) \in \mathtt{r}^{\mathbb{I}}\} \tag{6}$$
$$(\forall \mathtt{r}.\mathtt{C})^{\mathbb{I}} = \{x \in \Delta_{\mathbb{I}} \mid \forall y \in \Delta_{\mathbb{I}}, (x, y) \in \mathtt{r}^{\mathbb{I}} \implies y \in \mathtt{C}^{\mathbb{I}}\} \tag{7}$$

Notice how we are using FOL to express some of these equalities!

Example: illustration of $\exists \texttt{r}.\texttt{C}$

Example

Consider the concept

$$(\exists \texttt{hasComponent}.\texttt{Gripper})$$

that represents the set of all things that have at least one component that is a gripper. Given an interpretation $\mathbb{I}$,

$$(\exists \texttt{hasComponent}.\texttt{Gripper})^{\mathbb{I}} =$$
$$\{x \in \Delta_{\mathbb{I}} \mid \exists y \in \texttt{Gripper}^{\mathbb{I}}, (x, y) \in \texttt{hasComponent}^{\mathbb{I}}\} :$$

"The set of all elements x in $\Delta_{\mathbb{I}}$, such that there exists a gripper y such that x has a y for component".

Satisfiability of an $\mathcal{ALC}$ formula

Definition (Satisfiability)

Let F be an $\mathcal{ALC}$ formula, and $\mathbb{I}$ an interpretation. The fact that $\mathbb{I}$ *satisfies* F, denoted $\mathbb{I} \models F$, is given as follows depending on the structure of F:

- $\mathbb{I} \models \mathtt{C} \sqsubseteq \mathtt{D}$ if $\mathtt{C}^{\mathbb{I}} \subseteq \mathtt{D}^{\mathbb{I}}$; (if $F = \mathtt{C} \sqsubseteq \mathtt{D}$)
- $\mathbb{I} \models \mathtt{C} \equiv \mathtt{D}$ if $\mathtt{C}^{\mathbb{I}} = \mathtt{D}^{\mathbb{I}}$; (if $F = \mathtt{C} \equiv \mathtt{D}$)
- $\mathbb{I} \models \mathtt{C(a)}$ if $\mathtt{a}^{\mathbb{I}} \in \mathtt{C}^{\mathbb{I}}$; (if $F = \mathtt{C(a)}$)
- $\mathbb{I} \models \mathtt{r(a,b)}$ if $(\mathtt{a}^{\mathbb{I}}, \mathtt{b}^{\mathbb{I}}) \in \mathtt{r}^{\mathbb{I}}$; (if $F = \mathtt{r(a,b)}$)

Definition (Model)

An interpretation $\mathbb{I}$ is a *model* of a knowledge base KB, denoted $\mathbb{I} \models KB$, if $\mathbb{I}$ satisfies every axiom of KB.

From $\mathcal{ALC}$ to other DLs

- $\mathcal{ALC}$ is a small but expressive (it contains PL) description logic, introduced by Manfred Schmidt-Schauß and Gert Smolka in 1991.
- From it, we can build other DLs, e.g., by adding more constructs.
- This increases the expressiveness but also the complexity of the DL.
- To explore the consequences of adding more constructs, in particular in terms of the complexity of reasoning, you can visit
  http://www.cs.man.ac.uk/~ezolin/dl/
- The new DLs have names related to the contructs that are being added: for instance, $\mathcal{ALCQ}$ refers to the DL $\mathcal{ALC}$ augmented with "*Q*ualified number restriction" constructs.

In this section, we will briefly present some of the constructs we can add to $\mathcal{ALC}$, alongside some examples.

Nominals ($\mathcal{O}$: "$\mathcal{O}$ne of")

- We define a concept by *extension*, i.e., by explicitly indicating the elements in the concept:

$$\{\mathtt{E}_1,\ \mathtt{E}_2,\ \ldots,\ \mathtt{E}_n\}$$

Example

`{BlackColor, BlueColor, GreenColor, OrangeColor, RedColor, WhiteColor, YellowColor}`

Qualified number restriction ($\mathcal{Q}$)

- We have three additional constructs to express restrictions to the cardinality of roles (i.e. how many things are related to another one):
  - $(= n\ \mathtt{r}.\mathtt{C})$,
  - $(\leq n\ \mathtt{r}.\mathtt{C})$,
  - $(\geq n\ \mathtt{r}.\mathtt{C})$.

These constructs are interpreted as such:

- $(= n\ \mathtt{r}.\mathtt{C})^{\mathbb{I}} = \{x \in \Delta^{\mathbb{I}} \mid \mathrm{card}\{y \in \mathtt{C}^{\mathbb{I}} \mid (x, y) \in \mathtt{r}^{\mathbb{I}}\} = n\}$,
- $(\leq n\ \mathtt{r}.\mathtt{C})^{\mathbb{I}} = \{x \in \Delta^{\mathbb{I}} \mid \mathrm{card}\{y \in \mathtt{C}^{\mathbb{I}} \mid (x, y) \in \mathtt{r}^{\mathbb{I}}\} \leq n\}$,
- $(\geq n\ \mathtt{r}.\mathtt{C})^{\mathbb{I}} = \{x \in \Delta^{\mathbb{I}} \mid \mathrm{card}\{y \in \mathtt{C}^{\mathbb{I}} \mid (x, y) \in \mathtt{r}^{\mathbb{I}}\} \geq n\}$,

where $\mathrm{card}(X)$ indicates the number of elements of the set X.

Example

We can define the concept of all bipedal robots:

$$\mathtt{BipedalRobot} \equiv \mathtt{Robot} \sqcap (\mathtt{=2\ hasAppendage.Leg})$$

Role hierarchies ($\mathcal{H}$)

- We add the construct $\mathtt{r} \sqsubseteq \mathtt{s}$, where $\mathtt{r}$ and $\mathtt{s}$ are roles: $\mathtt{r}$ is *more specific* than $\mathtt{s}$.
- We can also use the construct $\mathtt{r} \equiv \mathtt{s}$ to indicate that the roles are "equivalent".

Example

- If a robot has an arm appendage, then in particular it also has an appendage:

$$\mathtt{hasArmAppendage} \sqsubseteq \mathtt{hasAppendage}$$

- A robot grasping something is also carrying something (we assume that the robot can only carry things by grasping them, and do not allow things like balancing a box on one's head):

$$\mathtt{carries} \equiv \mathtt{grasps}$$

Next week: OWL

- OWL (*W*eb *O*ntology *L*anguage) is a standard formalism and semantic basis for ontologies.
- It is built upon $\mathcal{ALC}$, with new constructs added to make it more descriptive, such as indications about roles, given in an RBox.
- Next week, you will learn about ontologies, using description logics such as OWL DL.