

# Knowledge Representation and Reasoning

Jef Wijsen

UMONS

February 4, 2026

# Table of Contents

- 1 Content and Methodology of the Course
- 2 Illustration of Automated Reasoning
- 3 Illustration of Solving
- 4 Recalls from Bases de Données I & II

$$\text{AI} \simeq \text{GOF AI} + \text{AML}$$

- AI: Artificial Intelligence
- GOF AI: Good Old-Fashioned Artificial Intelligence ( $\simeq$  symbolic AI)
- AML: Adaptive Machine Learning (reinforcement learning, big data...)

$$\text{KRR} \subseteq \text{GOF AI}$$

- KRR: Knowledge Representation and [Automated] Reasoning

# From My Conversation with ChatGPT (Feb. 2026)

Is there a future for KRR in LLMs?

Yes — **very much so**, and arguably even more than for classical automated reasoning.

In fact:

Knowledge Representation and Reasoning (KRR) is likely to become one of the main “structural backbones” of future LLM-based systems.

Let's look at why, and how.

- Web Ontology Language (**OWL**): logics that allow automated reasoning about data on the Web.

Task: **Automated reasoning**

Input: A set of logic formulas  $\Sigma$ ; a logic formula  $\sigma$ .

Question: Is  $\sigma$  a logical consequence of  $\Sigma$ ?

Since automated reasoning is computationally impossible for full first-order logic, one uses less expressive logics (a.k.a. Description Logics).


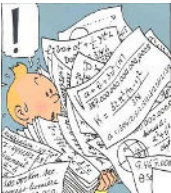

- Answer Set Programming (**ASP**): an expressive logic for specifying and solving problems in NP (including NP-complete problems).

Task: **Solving**

Input: A set  $\Sigma$  of logic formulas (also called constraints, rules... ).  
(e.g., the rules of Sudoku + a partially filled grid)

Question: Is there a solution (also called model, answer set... ) that satisfies every formula in  $\Sigma$ ?

# Course Methodology

	Classical course	↔	This course
language	French	↔	English
teacher's role	teaching	↔	guiding
students' role			 
	being taught	↔	scientific discovery tour
evaluation	exam	↔	project + homeworks + written exam

# Course Schedule

Just a screenshot (the full schedule is online):

This document may be updated during the course.

I will be abroad during the sessions marked with a superscript \*.

1	Wed, Feb. 4 (15H45)	Classroom meeting + <a href="#">organization (14')</a>
2	Thu, Feb. 5 (15H45)	<a href="#">motivation (72')</a>
4	Tue, Feb. 10 (15H45)	Classroom meeting; start Homework 1 (due on Feb. 23)
3	Wed, Feb. 11 (15H45)	<a href="#">introduction (170')</a>
5	Wed, Feb. 18 (15H45)	Classroom meeting
6	Thu, Feb. 19 (15H45)	<a href="#">modeling (106')</a>
7	Wed, Feb. 25 (15H45)	Classroom meeting; discuss Homework 1; start Homework 2 (due on March 9)
8	Thu, Feb. 26 (15H45)	<a href="#">language (128')</a>
9*	Tue, March 3 (15H45)	
10*	Wed, March 4 (15H45)	
11	Wed, March 11 (15H45)	Classroom meeting; discuss Homework 2; start Homework 3 (due on March 29);

# Table of Contents

- 1 Content and Methodology of the Course
- 2 Illustration of Automated Reasoning**
- 3 Illustration of Solving
- 4 Recalls from Bases de Données I & II

The **Entscheidungsproblem** was raised by Hilbert and Ackermann in 1928. It can be stated as follows:

*Is there an algorithm to determine whether a sentence  $\varphi$  of **first-order logic** is logically valid (true in all structures, finite or infinite)?*

In 1936, Church and Turing showed that the answer to this question is “no.” In 1950, Trakhtenbrot showed that the answer remains “no” when restricted to finite structures (as studied in finite model theory and database theory).

## Knowledge Base

A Description Logic (DL) knowledge base is a pair  $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ , where

- $\mathcal{T}$  is the **TBox**: terminological axioms
- $\mathcal{A}$  is the **ABox**: assertions about individuals

TBox:

Man	$\sqsubseteq$	Person
Woman	$\sqsubseteq$	Person
Man $\sqcap$ Woman	$\sqsubseteq$	$\perp$
Parent	$\equiv$	Person $\sqcap$ $\exists$ hasChild.Person
Father	$\equiv$	Man $\sqcap$ Parent
Mother	$\equiv$	Woman $\sqcap$ Parent

ABox:

Man(john)  
Woman(mary)  
hasChild(john, mary)

## Automated Reasoning

Given a **knowledge base**  $\mathcal{K}$  and an assertion  $\alpha$ , decide whether

$$\mathcal{K} \models \alpha.$$

For example,

- **Consistency checking:** Does  $\mathcal{K} \not\models \perp$ ?
- **Instance checking:** Does  $\mathcal{K} \models \text{Father}(\text{john})$ ?
- **Classification:** Does  $\mathcal{K} \models \text{Father} \sqsubseteq \text{Parent}$ ?
- **Query answering:** Find all individuals  $x$  s.t.  $\mathcal{K} \models \text{Parent}(x)$

**Description Logics** are designed to make these reasoning tasks decidable.

# Translation in First-Order Logic

$\forall x (\text{Man}(x) \rightarrow \text{Person}(x))$

$\forall x (\text{Woman}(x) \rightarrow \text{Person}(x))$

$\forall x \neg(\text{Man}(x) \wedge \text{Woman}(x))$

$\forall x \left( \text{Parent}(x) \leftrightarrow (\text{Person}(x) \wedge \exists y (\text{hasChild}(x, y) \wedge \text{Person}(y))) \right)$

$\forall x (\text{Father}(x) \leftrightarrow (\text{Man}(x) \wedge \text{Parent}(x)))$

$\forall x (\text{Mother}(x) \leftrightarrow (\text{Woman}(x) \wedge \text{Parent}(x)))$

$\text{Man}(\text{john})$

$\text{Woman}(\text{mary})$

$\text{hasChild}(\text{john}, \text{mary})$

We only need two variables,  $x$  and  $y$ . It is known that automated reasoning is possible in first-order logic restricted to two variables. This **two-variable fragment** forms the backbone of many Description Logics.

# Limits of the Two-Variable Fragment

$$\text{GrandParent} \equiv \text{Person} \sqcap \exists \text{hasChild} . (\exists \text{hasChild} . \text{Person})$$
$$\forall x (\text{GrandParent}(x) \leftrightarrow \exists y (\text{hasChild}(x, y) \wedge \exists x (\text{hasChild}(y, x))))$$

The following correct first-order definition requires three variables and cannot be expressed in the two-variable fragment:

$$\forall x \forall y (\text{hasGrandchild}(x, y) \leftrightarrow \exists z (\text{hasChild}(x, z) \wedge \text{hasChild}(z, y)))$$

One of the most striking “cliffs” in computational logic: While the two-variable fragment of first-order logic is decidable, adding just one more variable makes the logic expressive enough to simulate complex systems, leading to undecidability.

# Table of Contents

- 1 Content and Methodology of the Course
- 2 Illustration of Automated Reasoning
- 3 Illustration of Solving**
- 4 Recalls from Bases de Données I & II

# Solving Problems in NP

We are given a finite domain of **vertices** and a binary relation  $E$  of **edges**.

The following formula asks whether there exist three unary relations (i.e., three sets)  $A, B, C$  satisfying a **first-order condition**:

$$\exists A \exists B \exists C \left( \begin{array}{l} \forall x \left[ \begin{array}{l} (A(x) \wedge \neg B(x) \wedge \neg C(x)) \\ \vee (\neg A(x) \wedge B(x) \wedge \neg C(x)) \\ \vee (\neg A(x) \wedge \neg B(x) \wedge C(x)) \end{array} \right] \\ \wedge \\ \forall x \forall y \left( E(x, y) \rightarrow \neg \left[ \begin{array}{l} (A(x) \wedge A(y)) \\ \vee (B(x) \wedge B(y)) \\ \vee (C(x) \wedge C(y)) \end{array} \right] \right) \end{array} \right)$$

This is similar to querying relational databases, except that the logic used here is more expressive than relational calculus, which cannot express quantification over relations such as  $\exists A \exists B \exists C$ .

Assume a finite set of edge-facts, for example:

```
edge(1,2). edge(1,3). edge(2,3).
```

The following program finds all 3-colorings:

```
adjacent(X,Y) :- edge(X,Y).
```

```
adjacent(X,Y) :- edge(Y,X).
```

```
vertex(X) :- adjacent(X,Y).
```

```
green(X) :- vertex(X), not red(X), not blue(X).
```

```
blue(X) :- vertex(X), not red(X), not green(X).
```

```
red(X) :- vertex(X), not blue(X), not green(X).
```

```
:- adjacent(X,Y), red(X), red(Y).
```

```
:- adjacent(X,Y), blue(X), blue(Y).
```

```
:- adjacent(X,Y), green(X), green(Y).
```

# Table of Contents

- 1 Content and Methodology of the Course
- 2 Illustration of Automated Reasoning
- 3 Illustration of Solving
- 4 Recalls from Bases de Données I & II

- The **data complexity** of a query  $\{x_1, x_2, \dots, x_n \mid q(x_1, x_2, \dots, x_n)\}$  is the complexity of the following problem:

## Data Complexity

**INPUT:** A database instance  $I$ ; constants  
 $c_1, c_2, \dots, c_n$ .

**QUESTION:** Does  $I$  satisfy  $q(c_1, c_2, \dots, c_n)$ ?

- In many contexts, when we talk about a query language (e.g., Datalog), we implicitly refer to the set of all queries expressible in that language.
- For example, we say, “Datalog is in P (for data complexity)”, meaning that every query expressible in Datalog has data complexity in P.

# Data Complexity Classes

For **data complexity**, the following inclusions hold true:

$$\text{FO} \subsetneq \text{Datalog}^\neg \subseteq \text{P} \subseteq \text{NP} \subseteq \text{Prolog}$$

where

- FO denotes the class of problems that take as input a relational database instance and can be solved by a query in relational calculus; and
- $\text{Datalog}^\neg$  denotes the class of problems that take as input a relational database instance and can be solved by a program in Datalog with stratified negation.

Recall:

- NP-complete problems cannot be programmed in  $\text{Datalog}^\neg$ .
- Automated reasoning is already computationally impossible for FO.

- The **query complexity of a query language  $\mathcal{L}$**  is the complexity of the following problem, relative to a fixed database instance  $I$ :

## Query Complexity

**INPUT:** A query  $q(x_1, x_2, \dots, x_n)$  in  $\mathcal{L}$ ; constants  $c_1, c_2, \dots, c_n$ .

**QUESTION:** Does  $I$  satisfy  $q(c_1, c_2, \dots, c_n)$ ?

Recall:

- The query complexity of the class of conjunctive queries is already NP-complete.

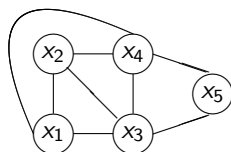
# Query Complexity is NP-Complete for Conjunctive Queries

## Fixed database instance

$b, g, r$  are three distinct constants, representing three colors.

$r$	$C_1$	$C_2$
	$b$	$g$
	$g$	$b$
	$b$	$r$
	$r$	$b$
	$g$	$r$
	$r$	$g$

## Query



$$q : \text{Answer}() \leftarrow R(x_1, x_2), R(x_2, x_1), \dots, \\ R(x_4, x_5), R(x_5, x_4)$$

Whenever there is an edge between  $x_i$  and  $x_j$  in the graph, the body of  $q$  contains  $R(x_i, x_j)$  and  $R(x_j, x_i)$ .

## Claim 1

$q(r) = \{\text{Answer}()\} \iff$  the graph encoded by  $q$  is 3-colorable.

See “A Datalog Primer.”

<https://web.umons.ac.be/app/uploads/sites/84/2024/06/primerDatalog.pdf>

# Datalog<sup>¬</sup> by Example

```
red(a,b). red(b,c). red(c,a).  
blue(a,c). blue(c,d). blue(d,a).  
redTrans(X,Y) :- red(X,Y).  
redTrans(X,Z) :- redTrans(X,Y), red(Y,Z).  
blueMonopoly(X,Y) :- blue(X,Y), not redTrans(X,Y).
```

- redTrans and blueMonopoly are **IDB** predicates (because they occur in rule heads); the other predicates are **EDB** predicates (= stored database relations).
- The **PDG** (Program Dependence Graph) has a (non-negated) edge from redTrans to redTrans, and a negated edge from blueMonopoly to redTrans.
- **Stratified semantics**: execute the rules for redTrans until no more redTrans-facts can be derived; only then can rules with “not redTrans” be evaluated.

# ASP by Example

```
person(john).  
happy(X) :- person(X), not unhappy(X).  
unhappy(X) :- person(X), not happy(X).
```

**Not stratified:** the 1st rule should be executed before the 2nd rule (because of “not happy”), but the 2nd should be executed before the 1st (because of “not unhappy”).

An ASP solver will find two models:

```
clingo version 4.5.4  
Solving...  
Answer: 1  
person(john) happy(john)  
Answer: 2  
person(john) unhappy(john)  
SATISFIABLE
```

Models : 2