Rule Mining in Knowledge Graphs

January 24, 2025

Motivation for Knowledge Graphs

The vision of Semantic Web : enable algorithms to interpret, reason, and interact intelligently with information.

How to represent knowledge in a form that can be processed by machines?

How to represent knowledge?

Relational Model:

- Well-defined structure, efficient for tabular data.
- Typically used for restrained number of relations.
- However, real-life relations are in the order of tens of thousands.

RDF: Resource Description Framework

- A standard proposed by the W3C to structure knowledge.
- Flexible and extensible format, suited for open and interconnected systems.

Principle of the RDF Model

Describe knowledge as elementary assertions:

Subject — Predicate — Object

Example:

- `Marie Curie` **was born in** `Warsaw`.
- `Marie Curie` **was married to** `Pierre Curie`.
- `Marie Curie` **lived in** `Paris`.
- `Marie Curie` **is a** `scientist`.
- `A scientist` **is a** `person`.

Simple model, but several considerations must be addressed: ambiguities, consistency, and predicate interoperability.

Addressing ambiguity

Representing knowledge with unique identifiers (IRIs) :

- For **predicates** (relations).
- For **entities** (people, concepts, places).

Addressing ambiguity

Representing knowledge with unique identifiers (IRIs) :

- For **predicates** (relations).
- For **entities** (people, concepts, places).

In Turtle, our examples can be expressed as follows :

```
@prefix yago:  <http://yago-knowledge.org/resource/>.
@prefix schema:  <https://schema.org/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
```

```
yago:Marie_Curie   schema:birthPlace   yago:Warsaw.
yago:Marie_Curie   schema:marriedTo    yago:Pierre_Curie.
yago:Marie_Curie   schema:livedIn      yago:Paris.
yago:Marie_Curie   rdf:type            yago:Scientist.
yago:Scientist     rdfs:subclassOf     yago:Person.
```

Textual descriptions: multilingual and multi-type

- Motivation: Some values need to be represented directly.
- A **literal** is a textual or numerical value considered to be fixed.
- A literal may be associated with a **language tag** or a **data type**.

Example :

`yago:Marie_Curie`	**rdfs:label**	`"Marie Curie"@fr.`
`yago:Marie_Curie`	**rdfs:label**	`"Maria Skłodowska-Curie"@pl.`
`yago:Marie_Curie`	**schema:birthDate**	`"1867-11-07"^^xsd:date.`

Note : The predicate `rdfs:label` is generally used to associate an entity with its **textual label** :

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

Knowledge Graphs Overview

Intuitively, a knowledge graph is a set of *triples*:

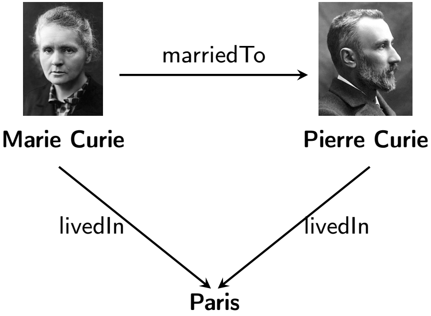
Subject — Predicate — Object

Definition: Let the following sets be given:

- $\mathcal{E}$: the set of entities (`IRI`).
- $\mathcal{P}$: the set of predicates (`IRI`).
- $\mathcal{L}$: the set of literal values (e.g., strings, numbers).

A knowledge graph is a set $\mathcal{K} \subseteq (\mathcal{E} \cup \mathcal{P}) \times \mathcal{P} \times (\mathcal{L} \cup \mathcal{E})$.

Simplified Graph Representation



Simplifications for readability:

- We omit the prefixes of `IRI` identifiers.
- We use images for some entities instead of their `IRI`s.
- We do not explicitly represent the labels (`rdf:label`).

Rules and Integrity Constraints

Examples of natural language expressions:

- Married people live in the same city.
- A person has only one birth date.
- Every subject of the predicate `marriedTo` is of type `Person`.
- Two authors of the same publication with the same name are the same person.

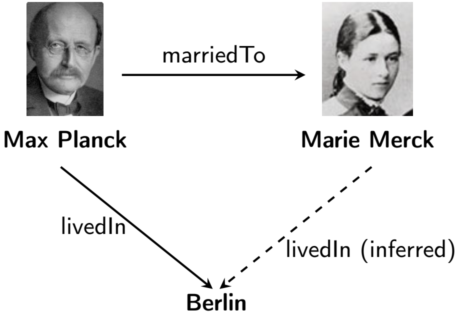
Rules and Integrity Constraints

Examples of natural language expressions:

- Married people live in the same city. *(Horn rule)*
- A person has only one birth date. *(functional dependency)*
- Every subject of the predicate `marriedTo` is of type `Person`. *(inclusion dependency)*
- Two authors of the same publication with the same name are the same person. *(key constraint)*

Applications of Rules: Fact Prediction

Rule: Married people live in the same locality.



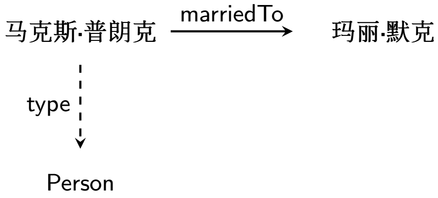
Type Inference

Rule: Every subject of the `marriedTo` relation is of type `Person`.

$$\text{马克斯·普朗克} \xrightarrow{\text{marriedTo}} \text{玛丽·默克}$$

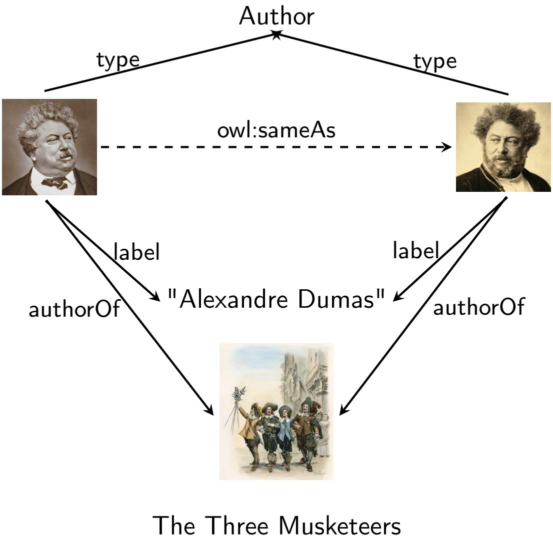
Type Inference

Rule: Every subject of the `marriedTo` relation is of type `Person`.



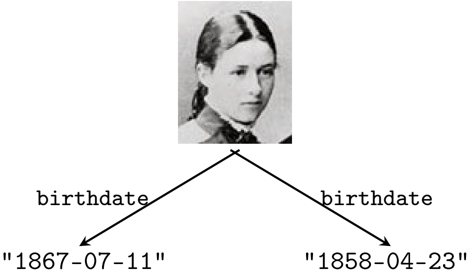
Inferring equivalent entities

Rule: Two authors sharing the same name and the same publications are the same person.



Reinforcing constraints

Rule: A person can have only one birthdate.



Exception raised: two birthdates for the same person.

Applications: detecting inconsistencies and cleaning data.

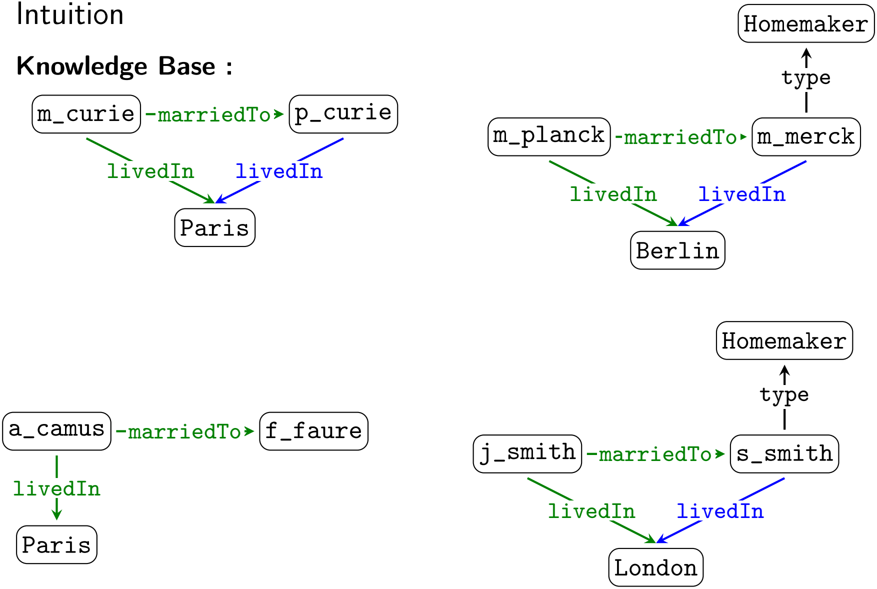
How to learn rules from Knowledge Graphs?

Induction approach: Generalise a number of similar observations into a hypothesis.

Example: Given many examples of spouses living together, generalise this knowledge.

Intuition

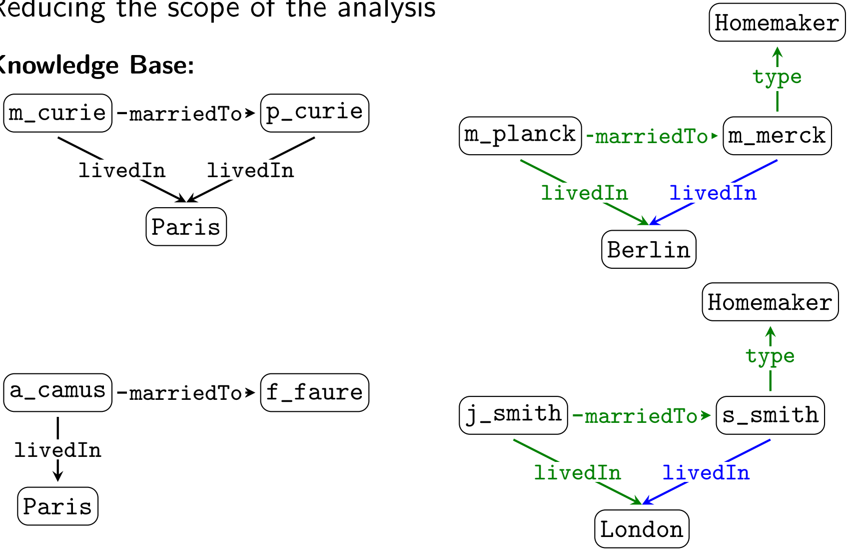
Knowledge Base :



Observation: In 75% of the cases, spouses are recorded as living together.

Reducing the scope of the analysis

Knowledge Base:



Observation: Reducing the analysis to spouses where one is a homemaker, the rule that the two live together holds in 100% of cases.

Challenges

- How to find the right balance between generality and specificity?

Challenges

- How to find the right balance between generality and specificity?
- How to evaluate the rules?

Challenges

- How to find the right balance between generality and specificity?
- How to evaluate the rules?
- Knowledge bases are incomplete.
  How can we distinguish between false and missing information?

Challenges

- How to find the right balance between generality and specificity?
- How to evaluate the rules?
- Knowledge bases are incomplete.
  How can we distinguish between false and missing information?
- The search space is too vast to cover efficiently.

Challenges

- How to find the right balance between generality and specificity?

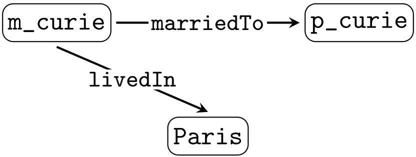
- How to evaluate the rules?
- Knowledge bases are incomplete.
  How can we distinguish between false and missing information?

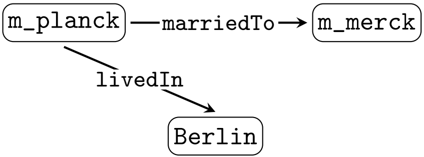
- The search space is too vast to cover efficiently.
- The search space depends on the expressivity of the rule language.

Challenges

- How to find the right balance between generality and specificity?
- How to evaluate the rules?
- Knowledge bases are incomplete.
  How can we distinguish between false and missing information?
- The search space is too vast to cover efficiently.
- The search space depends on the expressivity of the rule language.
- We need smart heuristics to uncover good rules.

Conjunctive queries for extracting frequent patterns





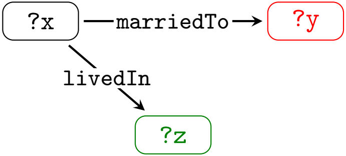
Query: Find all pairs (?y, ?z) such that the spouse of ?y lives in ?z.

The query in SPARQL :

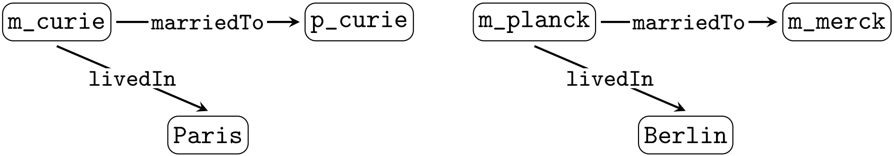
```
SELECT ?y ?z
WHERE {
  ?x marriedTo ?y.
  ?x livedIn ?z.
}
```

The query as a graph pattern:

select ?y, ?z



Conjunctive queries for extracting frequent patterns



Answers

?x	**?y**	**?z**
m_curie	p_curie	Paris
m_planck	m_merck	Berlin

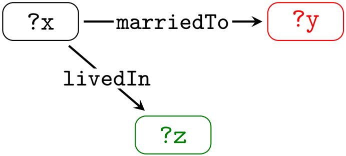
Query: Find all pairs (?y, ?z) such that the spouse of ?y lives in ?z.

The query in SPARQL :

```
SELECT ?y ?z
WHERE {
  ?x marriedTo ?y.
  ?x livedIn ?z.
}
```

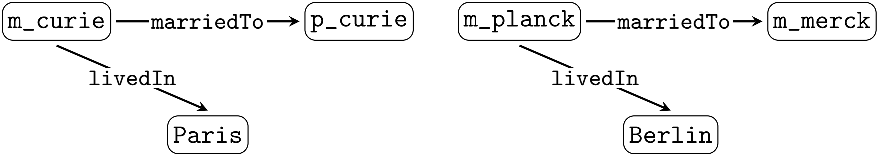
The query as a graph pattern:

select ?y, ?z



Datalog rewriting of the same query

Knowledge Graph:



Datalog rewriting of the same query

Knowledge Graph as a set of facts (grounded atoms):

```
marriedTo(m_curie, p_curie)      marriedTo(m_planck, m_merck)
    livedIn(m_curie, Paris)         livedIn(m_planck, Berlin)
```

Datalog rewriting of the same query

Knowledge Graph as a set of facts (grounded atoms):

```
marriedTo(m_curie, p_curie)      marriedTo(m_planck, m_merck)
    livedIn(m_curie, Paris)        livedIn(m_planck, Berlin)
```

Datalog Query:

$$\texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow R_1\texttt{(y, z)}$$

Datalog rewriting of the same query

Knowledge Graph as a set of facts (grounded atoms):

`marriedTo(m_curie, p_curie)` `marriedTo(m_planck, m_merck)`

`livedIn(m_curie, Paris)` `livedIn(m_planck, Berlin)`

Datalog Query:

$$\texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow R_1\texttt{(y, z)}$$

- `marriedTo(x, y)`, `livedIn(x, z)`, and R_1`(y, z)` are **atoms**.

Datalog rewriting of the same query

Knowledge Graph as a set of facts (grounded atoms):

`marriedTo(m_curie, p_curie)` `marriedTo(m_planck, m_merck)`

`livedIn(m_curie, Paris)` `livedIn(m_planck, Berlin)`

Datalog Query:

$$\texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow R_1\texttt{(y, z)}$$

- `marriedTo(x, y)`, `livedIn(x, z)`, and R_1`(y, z)` are **atoms**.
- The variables x, y, and z are placeholders for values from the KG.

Datalog rewriting of the same query

Knowledge Graph as a set of facts (grounded atoms):

`marriedTo(m_curie, p_curie)` `marriedTo(m_planck, m_merck)`

`livedIn(m_curie, Paris)` `livedIn(m_planck, Berlin)`

Datalog Query:

$$\texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow R_1\texttt{(y, z)}$$

- `marriedTo(x, y)`, `livedIn(x, z)`, and R_1`(y, z)` are **atoms**.
- The variables x, y, and z are placeholders for values from the KG.
- R_1 is the **query result relation**, generated from the rule.

Datalog rewriting of the same query

Knowledge Graph as a set of facts (grounded atoms):

`marriedTo(m_curie, p_curie)`	`marriedTo(m_planck, m_merck)`
`livedIn(m_curie, Paris)`	`livedIn(m_planck, Berlin)`

Datalog Query:

$$\texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow R_1\texttt{(y, z)}$$

- `marriedTo(x, y)`, `livedIn(x, z)`, and R_1`(y, z)` are **atoms**.
- The variables x, y, and z are placeholders for values from the KG.
- R_1 is the **query result relation**, generated from the rule.

Query Answers: Each valid **substitution** for x, y, and z that satisfies the rule generates a row in the result:

?x	**?y**	**?z**
m_curie	p_curie	Paris
m_planck	m_merck	Berlin

Example Rule R**:** `marriedTo(x,y), livedIn(x,z)` $\Rightarrow$ `livedIn(y,z)`

- A fact `p` predicted by the Knowledge Graph $\mathcal{K}$ and rule R is denoted:

$$\mathcal{K} \wedge R \models \mathtt{p}$$

  Here, $\models$ indicates logical entailment, meaning `p` is implied by $\mathcal{K}$ and R.

Introduction to Horn Rules

Example Rule R**:** `marriedTo(x,y), livedIn(x,z)` $\Rightarrow$ `livedIn(y,z)`

- A fact `p` predicted by the Knowledge Graph $\mathcal{K}$ and rule R is denoted:

$$\mathcal{K} \wedge R \models \mathtt{p}$$

  Here, $\models$ indicates logical entailment, meaning `p` is implied by $\mathcal{K}$ and R.
- The rule states: If a person x is married to y and x lived in a city z, then y is also inferred to have lived in z.

Example Rule R: `marriedTo(x,y), livedIn(x,z)` $\Rightarrow$ `livedIn(y,z)`

- A fact $\mathtt{p}$ predicted by the Knowledge Graph $\mathcal{K}$ and rule R is denoted:

$$\mathcal{K} \wedge R \models \mathtt{p}$$

  Here, $\models$ indicates logical entailment, meaning $\mathtt{p}$ is implied by $\mathcal{K}$ and R.
- The rule states: If a person x is married to y and x lived in a city z, then y is also inferred to have lived in z.
- The variables x, y, and z are **universally quantified**, meaning the rule applies to all individuals in $\mathcal{K}$.

Introduction to Horn Rules

Example Rule R**:** `marriedTo(x,y), livedIn(x,z)` $\Rightarrow$ `livedIn(y,z)`

- A fact `p` predicted by the Knowledge Graph $\mathcal{K}$ and rule R is denoted:

$$\mathcal{K} \wedge R \models \texttt{p}$$

  Here, $\models$ indicates logical entailment, meaning `p` is implied by $\mathcal{K}$ and R.
- The rule states: If a person x is married to y and x lived in a city z, then y is also inferred to have lived in z.
- The variables x, y, and z are **universally quantified**, meaning the rule applies to all individuals in $\mathcal{K}$.

Application to our KG:

- $\mathcal{K} \wedge R \models$ `livedIn(p_curie, Paris)`
- $\mathcal{K} \wedge R \models$ `livedIn(m_merck, Berlin)`

Mining Horn Rules: Reducing the Search Scope

- Systems like **AMIE** and **AnyBURL** focus on mining **positive Horn rules**:
  - The conclusion consists of a single positive literal.
  - The premise is a conjunction of positive literals.
- To further reduce the search space, they impose additional constraints:
  - **Connectedness:** All atoms in the rule must be transitively connected.
  - **Closedness:** A rule is closed if:
    - All variables appear in at least two atoms.
- A closed rule ensures **safely**: variables in conclusion appear in the premise.

Expressive Power

Question: Among the following formulas, which are not Horn rules?

- **Married people live in the same city.**
- **A person has only one date of birth.**
- **Every subject of the predicate** `marriedTo` **is of type** `Person`**.**
- **Two authors of the same publication with the same name are the same person.**

Expressive Power

Question: Among the following formulas, which are not Horn rules?

- **Married people live in the same city.**

  $\texttt{marriedTo}(x, y)\texttt{, livesIn}(x, z) \Rightarrow \texttt{livesIn}(y, z)$ ✓

- **A person has only one date of birth.**
- **Every subject of the predicate** `marriedTo` **is of type** `Person`**.**
- **Two authors of the same publication with the same name are the same person.**

Expressive Power

Question: Among the following formulas, which are not Horn rules?

- **Married people live in the same city.**

  `marriedTo`(x, y), `livesIn`(x, z) $\Rightarrow$ `livesIn`(y, z) ✓

- **A person has only one date of birth.**

  `birthdate`(x, d_1), `birthdate`(x, d_2) $\Rightarrow d_1 = d_2$ ✗

- **Every subject of the predicate `marriedTo` is of type `Person`.**

- **Two authors of the same publication with the same name are the same person.**

Expressive Power

Question: Among the following formulas, which are not Horn rules?

- **Married people live in the same city.**

  $\texttt{marriedTo}(x, y)\texttt{,}\ \texttt{livesIn}(x, z) \Rightarrow \texttt{livesIn}(y, z)$ ✓

- **A person has only one date of birth.**

  $\texttt{birthdate}(x, d_1)\texttt{,}\ \texttt{birthdate}(x, d_2) \Rightarrow d_1 = d_2$ ✗

- **Every subject of the predicate `marriedTo` is of type `Person`.**

  $\texttt{marriedTo}(x, y) \Rightarrow \texttt{type}(x\texttt{,}\ '\texttt{Person}')$ ✓

- **Two authors of the same publication with the same name are the same person.**

Expressive Power

Question: Among the following formulas, which are not Horn rules?

- **Married people live in the same city.**

  $\texttt{marriedTo}(x, y),\ \texttt{livesIn}(x, z) \Rightarrow \texttt{livesIn}(y, z)$ ✓

- **A person has only one date of birth.**

  $\texttt{birthdate}(x, d_1),\ \texttt{birthdate}(x, d_2) \Rightarrow d_1 = d_2$ ✗

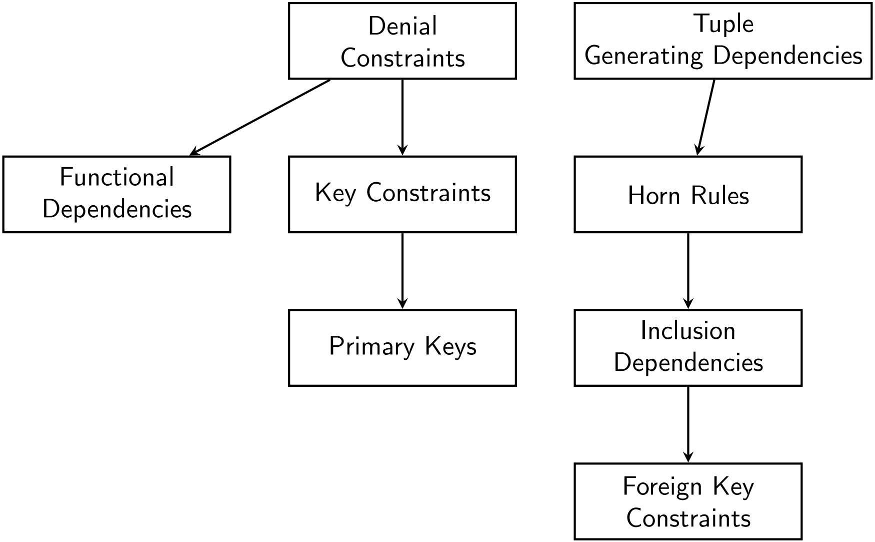
- **Every subject of the predicate `marriedTo` is of type `Person`.**

  $\texttt{marriedTo}(x, y) \Rightarrow \texttt{type}(x,\ '\texttt{Person}')$ ✓

- **Two authors of the same publication with the same name are the same person.**

  $\texttt{authorOf}(x, p),\ \texttt{authorOf}(y, p),\ \texttt{label}(x, n),\ \texttt{label}(y, n) \Rightarrow x = y$ ✗

Expressivity: Hierarchy of Languages



Lattice: Rule Refinements Exploration

Initial Query: $\Rightarrow$ `livesIn(y,z)`

$$\begin{tikzcd}
\Rightarrow \texttt{livesIn(y,z)} \arrow[d, "\textit{Add Dangling Atom}"'] \\
\texttt{marriedTo(x,y)} \Rightarrow \texttt{livesIn(y,z)} \arrow[d, "\textit{Add Closing Atom}"'] \\
R_1\text{: } \texttt{marriedTo(x,y), livedIn(x,z)} \Rightarrow \texttt{livesIn(y,z)} \arrow[d, "\textit{Add Instantiated Atom}"'] \\
R_2\text{: } \texttt{marriedTo(x,y), livedIn(x,z), type(x,'Homemaker')} \Rightarrow \texttt{livesIn(y,z)}
\end{tikzcd}$$

Query Containment

Rules can be generated by **extending existing rules**, forming a containment relationship between rules.

Example:

R_1 : `marriedTo(x,y), livedIn(x,z)` $\Rightarrow$ `livedIn(y,z)`

R_2 : `marriedTo(x,y), livedIn(x,z), type(x,'Homemaker')` $\Rightarrow$ `livedIn(y,z)`

- Does R_2 generate a subset of predictions compared to R_1?

Inclusion of Conjunctive Queries

Queries:

R_1 : `marriedTo(x, y), livedIn(x, z)` $\Rightarrow$ R_1`(y, z)`

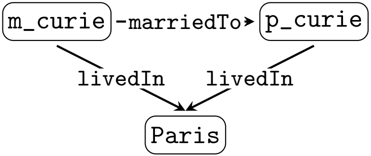
R_2 : `marriedTo(x, y), livedIn(x, z), type(x, 'Homemaker')` $\Rightarrow$ R_2`(y, z)`

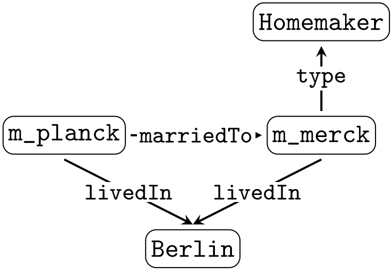
- Regardless of the knowledge base, the results of query R_2 are always included in those of query R_1.
- Formally, R_2 is a sub-query of R_1, denoted as:

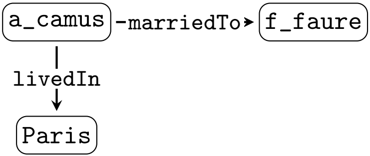
$$R_2 \sqsubseteq R_1$$

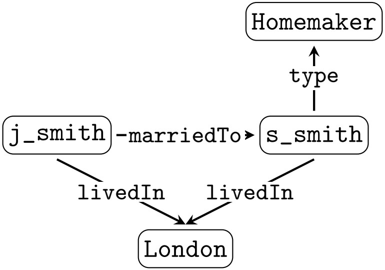
KG Example

Knowledge Base:









Specialization of Rules

The specialization of a rule can be achieved in two main ways:

- **Adding atoms** to the body of a rule (as seen previously).
- **Replacing variables with constants**, restricting the application of the rule to a specific subset.

Example:

$$R_1: \quad \texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow \texttt{livedIn(y, z)}$$

$$R_3: \quad \texttt{marriedTo(x, y), livedIn(x, }'\texttt{Paris}'\texttt{)} \Rightarrow \texttt{livedIn(y, }'\texttt{Paris}'\texttt{)}$$

Observation: R_3 applies only to people married to Parisians.

Specialization of Rules

The specialization of a rule can be achieved in two main ways:

- **Adding atoms** to the body of a rule (as seen previously).
- **Replacing variables with constants**, restricting the application of the rule to a specific subset.

Example: Let us compare the results of the two rules:

$$R_1 : \quad \texttt{marriedTo(x, y), livedIn(x, z)} \Rightarrow R_1\texttt{(y, z)}$$

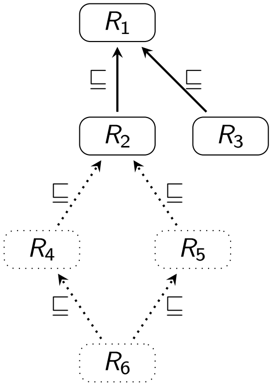
$$R_3 : \quad \texttt{marriedTo(x, y), livedIn(x, 'Paris')} \Rightarrow R_3\texttt{(y, 'Paris')}$$

Notice that here as well, $R_3 \sqsubseteq R_1$.

Observation: R_3 applies only to people married to Parisians.

Rule Generation Lattice

- Rule generation can be represented as a lattice, where rules are connected by the inclusion operator $\sqsubseteq$.
- General rules are at the top, while specific rules are at the bottom.
- Lattices optimize rule generation by factoring computations and avoiding redundancies.



Question: Where should rule specialization stop?

Challenge: Finding the right balance between generality and specificity.

Rule Support

Definition: The support of a rule R is the number of true facts it generates:

$$support(R) = |\{p : (\mathcal{K} \wedge R \models p) \wedge p \in \mathcal{K}\}|$$

Property: The support decreases as a rule becomes more specialized.

Utility: A rule is not refined if its support falls below a threshold.

Question: What support threshold should you choose for your data?

Calculating Rule Support

Rules:

R_1 : `marriedTo(x,y), livedIn(x,z)` $\Rightarrow$ `livedIn(y,z)`

R_2 : `marriedTo(x,y), livedIn(x,z), type(x,'Homemaker')` $\Rightarrow$ `livedIn(y,z)`

Predictions:

R_1

livedIn	
p_curie	Paris
m_merck	Berlin
f_faure	Paris
s_smith	London

R_2

livedIn	
m_merck	Berlin
s_smith	London

Confidence Measure of a Rule

Definition: The confidence of a rule R is the proportion of true predictions among all predictions:

$$confidence(R) = \frac{support(R)}{support(R) + |cex(R)|}$$

where $cex(R)$ represents the counterexamples of R.

Extension: AMIE 3 generalizes Inductive Logic Programming (ILP) by discovering *soft* rules that tolerate a limited number of counterexamples.

Summary:

- **Support:** Relevance of a rule.
- **Confidence:** Accuracy of a rule.

Objective: Extract rules with support and confidence above defined thresholds.

The Problem of Counterexamples

Observation: Knowledge bases primarily contain positive information but lack explicit counterexamples.

Solution: Negative facts can be inferred based on assumptions:

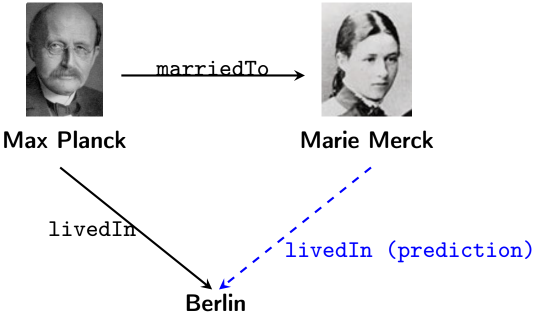
- **Closed World Assumption (CWA):** What is not known is false.
- **Open World Assumption (OWA):** What is not known is unknown (and could therefore be true).
- **Partial Completeness Assumption (PCA):** Proposed by AMIE.

Counterexample or True Prediction?

Consider the prediction:

$$\mathcal{K} \wedge R \models \texttt{livedIn(Marie Merck, Berlin)}$$

where $\mathcal{K}$ is the following knowledge base:



Interpretation based on assumptions:

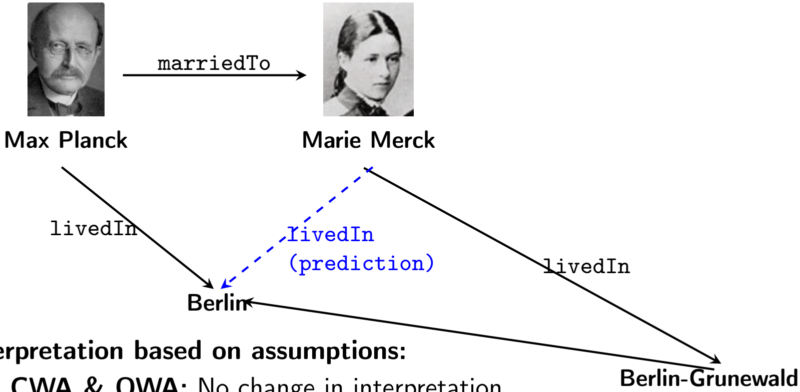
- **CWA:** The prediction is false, so it is a counterexample.
- **OWA:** No conclusion can be drawn; the fact might be unknown.
- **PCA:** The prediction is true because no other residence is known for `Marie Merck`.

Knowledge Base Update

Consider the prediction:

$$\mathcal{K}' \wedge R \models \texttt{livedIn(Marie Merck, Berlin)}$$

where $\mathcal{K}'$ is the following knowledge base:



Interpretation based on assumptions:

- **CWA & OWA:** No change in interpretation.
- **PCA:** The prediction is a counterexample, as another fact `livedIn(Marie Merck, Berlin-Grunewald)` is known.

Overview of Rule Discovery Methods

- **Inductive Logic Programming (ILP):** Finding hypotheses that cover examples.
- **Approaches:**
  - *Top-Down:* Based on specialization (e.g., AMIE, RUDIK).
  - *Bottom-Up:* Based on generalization (e.g., GOLEM, AnyBURL).
- **Interesting Perspective:**
  - Rule discovery on vector representations.

Bibliography

- Luis Galárraga, Christina Teflioudi, Katja Hose, Fabian M. Suchanek: *"AMIE: Association Rule Mining under Incomplete Evidence"* WWW 2013.
- N. Lavrač, S. Džeroski: *"Inductive Logic Programming — Techniques and Applications."* References.
- F. Suchanek, J. Lajus, A. Boschin, G. Weikum: *"Knowledge Representation and Rule Mining in Entity-Centric KBs"* Reasoning Web Summer School (RW), 2019.