Rule Mining in Knowledge Graphs

Pablo Mollá Chárlez

February 8, 2025

Contents

1 Rule Mining **2**
1.1 Challenges . . . 2
1.2 Techniques/Tools to Address Challenges . . . 2

2 Horn Rules **3**
2.1 Horn Rule Properties . . . 4
2.2 Rule Evaluation: Quality Measures . . . 5
2.2.1 The Problem of Counterexamples . . . 5
2.3 Overview of Rule Discovery Methods . . . 5

1 Rule Mining

In rule mining for knowledge graphs, we automatically discover logical statements—often called rules or integrity constraints -that capture regularities in the data. Below are four common types of rules/constraints, each illustrated by a natural-language example:

- Horn Rule: A Horn rule has the form Body $\Longrightarrow$ Head, meaning whenever the conditions in the "Body" are satisfied, the "Head" must also be true. For instance, "Married people live in the same city." Formally, if x and y are married, then x and y share a city.
- Functional Dependency: A functional dependency states that one property (or set of properties) uniquely determines another property. In relational databases, we often write this as $X \to Y$. For example, "A person has only one birth date." In other words, knowing the identity of the person uniquely determines that person's birth date.
- Inclusion Dependency: An inclusion dependency enforces that all instances falling under a certain property or relationship must also fit into a specified category. For instance, "Every subject of the predicate **marriedTo** is of type **Person**." Formally, if an entity is used in **marriedTo**, it must be included in the class **Person**.
- Key Constraint: A key constraint says that one or more properties constitute a unique identifier for entities, or that matching on these properties implies the entities are the same. For example, "Two authors of the same publication with the same name are the same person." Here, the combination of $\{\text{publication}, \text{name}\}$ effectively behaves as a key, identifying a unique individual.

By learning and enforcing such rules or constraints, one can improve the consistency, quality, and expressiveness of knowledge graphs.

1.1 Challenges

We can face the followingchallenges when searching or discovering rules in knowledge graphs:

1. **Generality vs. Specificity** Finding rules that are neither too broad (over-generalizing and capturing false statements) nor too narrow (missing key relationships) can be difficult.
2. **Evaluating the Rules** Determining how "good" or "correct" a rule is often tricky—especially when ground truth or labeled data is limited.
3. **Incomplete Knowledge Bases** Many KBs have missing facts, so a rule might appear false when the evidence is just unrecorded. Distinguishing genuine errors from missing information is a major hurdle.
4. **Vast Search Space** There are potentially countless candidate rules, and exhaustively exploring them all is impractical.
5. **Rule-Language Expressivity** The complexity of rules allowed (e.g., Horn clauses, existential quantifiers) further expands the search space and adds computational overhead.
6. **Smart Heuristics** Given these constraints, efficient heuristics and pruning strategies are critical for discovering high-quality rules without wasting time on unpromising candidates.

1.2 Techniques/Tools to Address Challenges

Conjunctive queries and Datalog rewritings don't by themselves solve the rule-mining challenges—rather, they are tools or techniques that can **help address** them by extracting patterns that can then be used or generalized into rules. For example:

- **Handling a huge search space:** Conjunctive queries (or their Datalog rewritings) let you systematically enumerate patterns (joins, conditions) in a more structured way, pruning unpromising candidates more easily.

- **Dealing with incomplete data:** A query-based approach can pinpoint exactly where data is missing or contradictory, and thus help differentiate "no match found" from "fact not present" in the knowledge base.
- **Ensuring the right level of specificity:** Queries can be refined (or rewritten in Datalog) to capture increasingly specific patterns, enabling a "drill-down" to discover rules with appropriate granularity.

So, rather than being complete solutions to the challenges, they serve as methodological steps or paths for organizing, pruning, and checking the rule or pattern search. For instance, conjunctive queries using SPARQL can be used to find patterns. By specifying a pattern like

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

we identify all matching triples in the graph. Repeated occurrences of the same pattern can point to a potential rule (e.g., "If $?x$ is married to $?y$, then $?x$ and $?y$ share a location"). On the other hand, datalog represents the same pattern as a logical rule, such as

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

This logical form allows easy composition with other rules, unification of variables, and automated reasoning (e.g., checking for contradictions or inferring additional facts).

2 Horn Rules

A Horn rule is a logical rule of the form:

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \ \Rightarrow\ B,$$

where A_i and B are positive literals (like $\text{marriedTo}(x, y)$), and the variables are universally quantified—they apply to all individuals in the knowledge base. For instance, a horn rule might be:

$$\text{marriedTo}(x, y),\ \text{livedIn}(x, z) \ \Rightarrow\ \text{livedIn}(y, z).$$

This states that for any x, y, and z, if x is married to y and x lived in z, then y also lived in z. In logical notation, the knowledge graph K plus the rule R entails the fact p:

$$K \ \wedge\ R \ \vDash\ p.$$

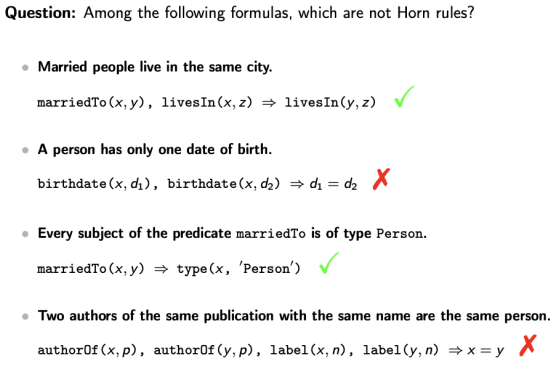

Figure 1: Horn Rules Examples

2.1 Horn Rule Properties

Systems like **AMIE** and **AnyBURL** mine horn rules from knowledge graphs. They focus on positive Horn rules - where both the premise (the body) and the conclusion (the head) consist only of positive atoms. To reduce the enormous search space, they often impose additional constraints such as:

- **Connectedness:** A rule is connected if all atoms (premise plus conclusion) share variables in such a way that you can travel from any variable to any other via those shared variables. For instance,

$$\text{knows}(x, y),\ \text{livesIn}(y, z) \ \Rightarrow\ \text{friendOf}(x, z).$$

  Here, $x \to y \to z$ forms a single chain: we can "connect" all variables x, y, z. On the contrary,

$$\text{knows}(x, y),\ \text{likes}(u, v) \ \Rightarrow\ \text{friendOf}(x, z).$$

  The atom $\text{likes}(u, v)$ is disjoint from the other atoms; there is no path of shared variables bridging (x, y) and (u, v).

- **Closedness:** A rule is closed if (1) every variable appears in at least two atoms, and (2) variables used in the conclusion also appear in the premise (this property is called **safely**). For example, a closed example could be:

$$\text{marriedTo}(x, y),\ \text{livesIn}(x, z) \ \Rightarrow\ \text{livesIn}(y, z).$$

  - x appears in $\text{marriedTo}(x, y)$ and $\text{livesIn}(x, z)$.
  - y appears in $\text{marriedTo}(x, y)$ and $\text{livesIn}(y, z)$.
  - z appears in $\text{livesIn}(x, z)$ and $\text{livesIn}(y, z)$.

  Because all three variables show up at least twice—and z doesn't magically appear only in the conclusion - this rule is closed. On the contrary,

$$\text{knows}(x, y) \ \Rightarrow\ \text{friendOf}(x, z).$$

  Here, the variable z appears only in the conclusion ($\text{friendOf}(x, z)$) and never in the premise. That makes it not closed (and also not safe for many logic engines).

- **Query Containment:** Horn rules can be "extended" by adding more atoms in the premise (the body). For example,

$$R1 : \mathit{marriedTo}(x, y),\ \mathit{livedIn}(x, z) \ \Rightarrow\ \mathit{livedIn}(y, z)$$

$$R2 : \mathit{marriedTo}(x, y),\ \mathit{livedIn}(x, z),\ \mathit{type}(x, \text{'Homemaker'}) \ \Rightarrow\ \mathit{livedIn}(y, z).$$

  Here, $R2$ is contained in $R1$, meaning any instance satisfying $R2$ also satisfies $R1$. Formally, $R2 \subseteq R1$.

- **Rule Specialization:** Extending a rule by adding atoms to the body or by replacing variables with constants is called specialization. As a rule becomes more specialized, it applies to fewer (more specific) cases-so its support typically goes down, while its precision may go up. For instance,

$$R3 :\ \mathit{marriedTo}(x, y),\ \mathit{livedIn}(x, \text{'Paris'}) \ \Rightarrow\ \mathit{livedIn}(y, \text{'Paris'})$$

  This only triggers if x lives in Paris, restricting the scope of $R1$. We can view rules in a lattice structure where general rules sit at the top and specific rules appear lower down. Moving **from top to bottom** corresponds to specialization—adding more conditions or constraints (this procedurea and structure is called Rule Generation Lattice)

2.2 Rule Evaluation: Quality Measures

- Support (Relevance)

$$\text{support}(R) = |\{p : (K \wedge R \vDash p) \wedge p \in K\}|$$

  Measures how many facts in K the rule correctly predicts, and it decreases as the rule becomes more specialized (fewer cases match).

- Confidence (Accuracy)

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

  Where $\text{cex}(R)$ is the set of counterexamples-cases that the rule predicts but are actually false. Confidence goes up when the rule has fewer counterexamples relative to the number of correct predictions.

2.2.1 The Problem of Counterexamples

Many knowledge graphs are incomplete: they rarely store explicit negative facts. Therefore, systems often rely on assumptions like:

- Closed World Assumption (CWA): anything not stated is false.
- Open World Assumption (OWA): anything not stated is unknown.
- Partial Completeness Assumption (PCA): a middle ground (used by AMIE) to infer negative facts more carefully.

2.3 Overview of Rule Discovery Methods

Both **top-down** and **bottom-up** search strategies are part of Inductive Logic Programming (ILP). ILP is a broad framework that can be instantiated in different ways:

- **Top-down ILP** (e.g., AMIE, RUDIK) starts with very general rules and specializes them by adding conditions.
- **Bottom-up ILP** (e.g., GOLEM, AnyBURL) begins with specific examples or "clauses" and generalizes them step by step.

So these two approaches are both part of the ILP family—just different ways of navigating the search space for logic rules. Together, these techniques (query containment, specialization, support/confidence measures, etc.) build on the Horn rule framework and the earlier constraints (connectedness, closedness), providing a structured way to navigate and prune the huge space of possible rules.