


# DEER

## Automating RDF Dataset Transformation and Enrichment

Mohamed Ahmed Sherif, Axel-Cyrille Ngonga Ngomo and Jens Lehmann

June 3, 2015

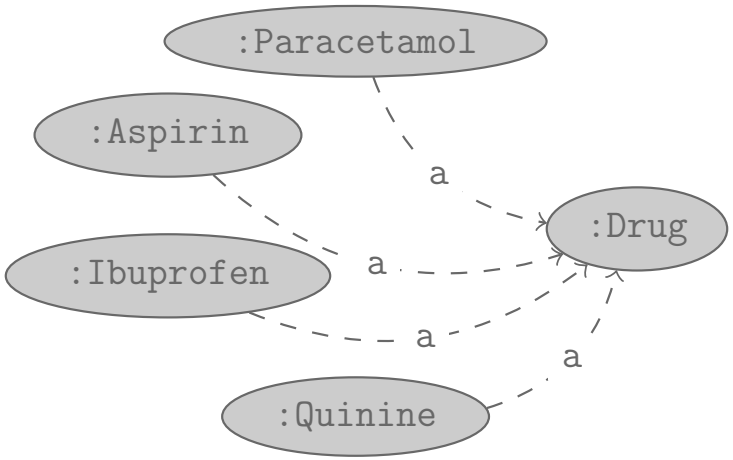
# Outline

# Outline

# Why RDF Transformation & Enrichment?

**Dataset** *DrugBank*

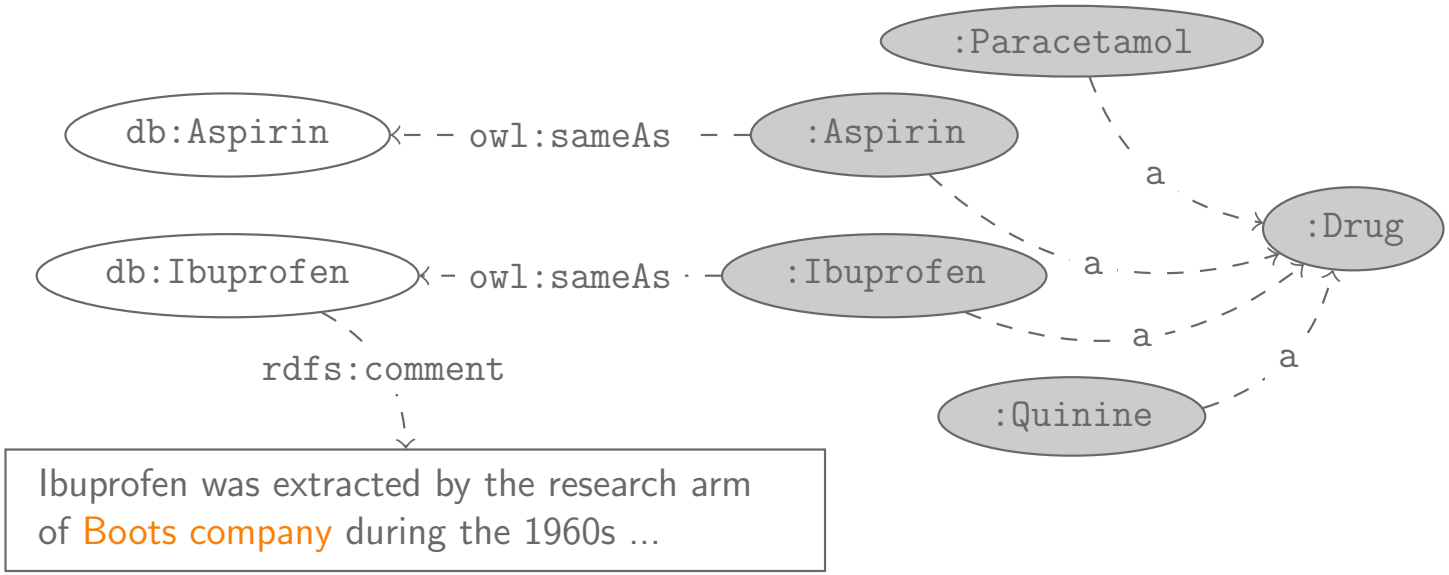
**Goal** Gather information about companies related to drugs for a market study



# Why RDF Transformation & Enrichment?

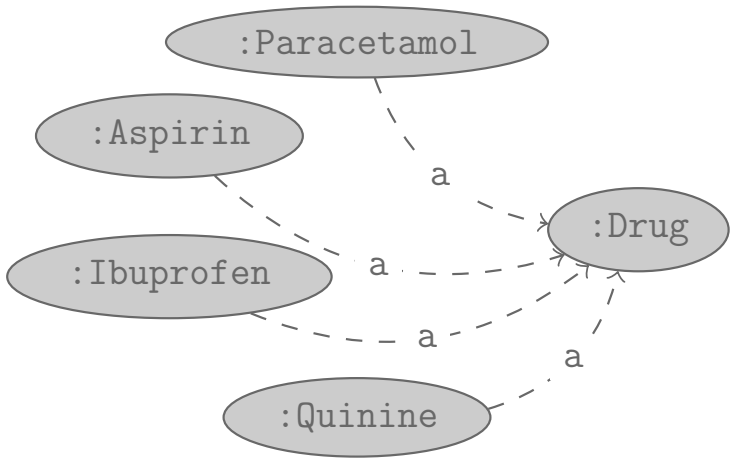
**Dataset** *DrugBank*

**Goal** Gather information about companies related to drugs for a market study



# RDF Transformation & Enrichment

- Need for enriched datasets
  - Tourism
  - Question Answering
  - Enhanced Reality
  - ...
- RDF transformation and enrichment
  - Triples to be added to the original KB and/or
  - Triples to be deleted from the original KB



# Manual Knowledge Base Enrichment

- Demands for the specification of data *enrichment pipelines*
- Describe how data is to be integrated (usually manually)

**Manual customized enrichment pipelines**

- ⊕ Leads to the expected results
- ⊖ Time consuming
- ⊖ Cannot be ported easily to other datasets

# Manual Knowledge Base Enrichment

- Demands for the specification of data *enrichment pipelines*
- Describe how data is to be integrated (usually manually)

**Manual customized enrichment pipelines**

- ⊕ Leads to the expected results
- ⊖ Time consuming
- ⊖ Cannot be ported easily to other datasets

# Automatic Knowledge Base Enrichment

- *Enrichment pipeline* $M : \mathcal{K} \rightarrow \mathcal{K}$ that maps KB $K$ to an enriched KB $K'$ with $K' = M(K)$.
- $M$ is an ordered list of *atomic enrichment functions* $m \in \mathcal{M}$

$$M = \begin{cases} \phi & \text{if } K = K', \\ (m_1, \ldots, m_n), \text{where } m_i \in \mathcal{M}, 1 \leq i \leq n & \text{otherwise.} \end{cases}$$

## Research questions

1. How to create self-configuring atomic enrichment functions $m \in \mathcal{M}$?
2. How to automatically generate an enrichment pipeline $M$?

# Automatic Knowledge Base Enrichment

- *Enrichment pipeline* $M : \mathcal{K} \rightarrow \mathcal{K}$ that maps KB $K$ to an enriched KB $K'$ with $K' = M(K)$.
- $M$ is an ordered list of *atomic enrichment functions* $m \in \mathcal{M}$

$$M = \begin{cases} \phi & \text{if } K = K', \\ (m_1, \ldots, m_n), \text{where } m_i \in \mathcal{M}, 1 \leq i \leq n & \text{otherwise.} \end{cases}$$

## Research questions

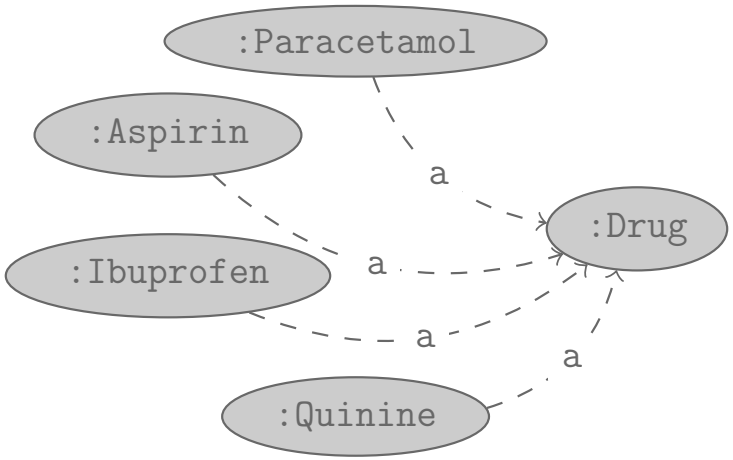
1. How to create self-configuring atomic enrichment functions $m \in \mathcal{M}$?
2. How to automatically generate an enrichment pipeline $M$?

# Outline

# Atomic Enrichment Functions

## I. Dereferencing atomic enrichment function

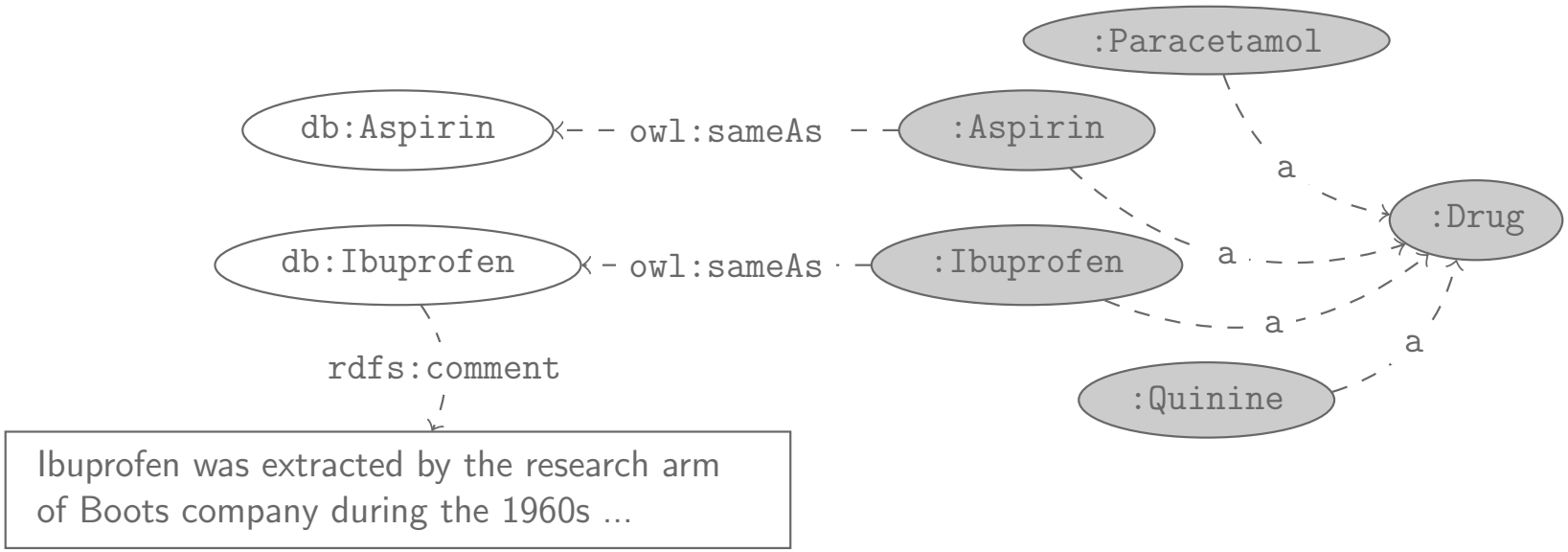
- Datasets are linked (e.g., using `owl:sameAs`)
- Deferences pre-specified set of predicates
- Adds found predicates to source the dataset



# Atomic Enrichment Functions

## I. Dereferencing atomic enrichment function

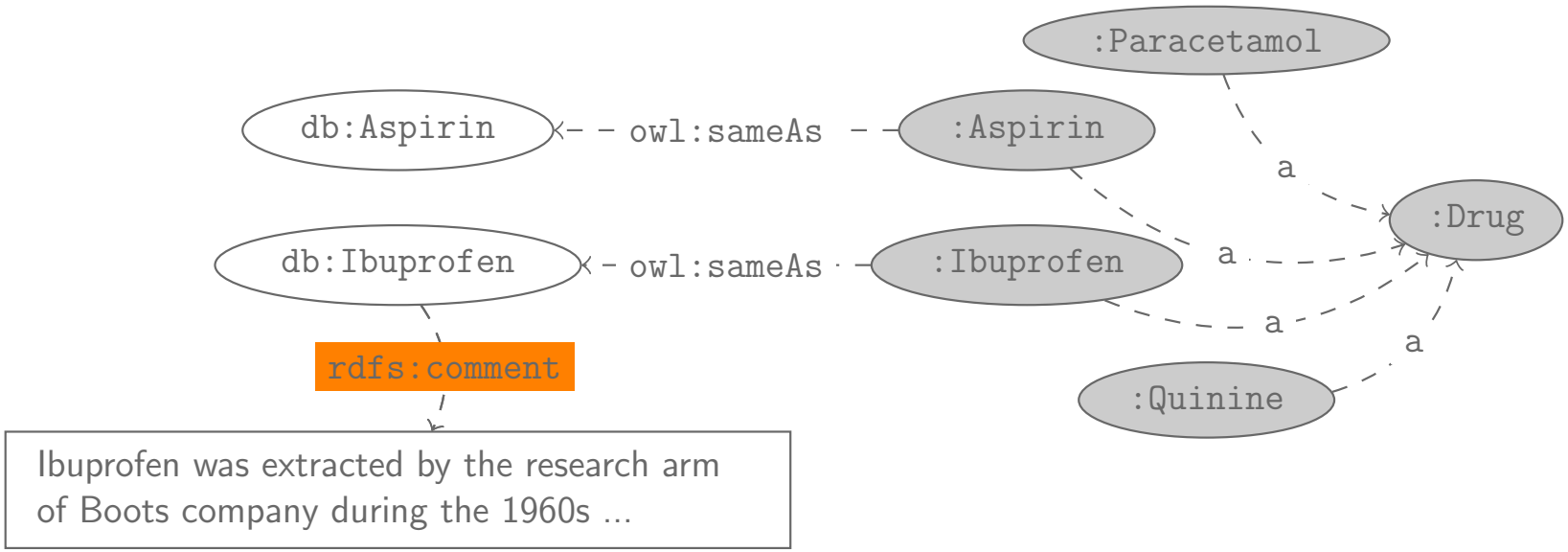
- Datasets are linked (e.g., using `owl:sameAs`)
- Deferences pre-specified set of predicates
- Adds found predicates to source the dataset



# Atomic Enrichment Functions

## I. Dereferencing atomic enrichment function

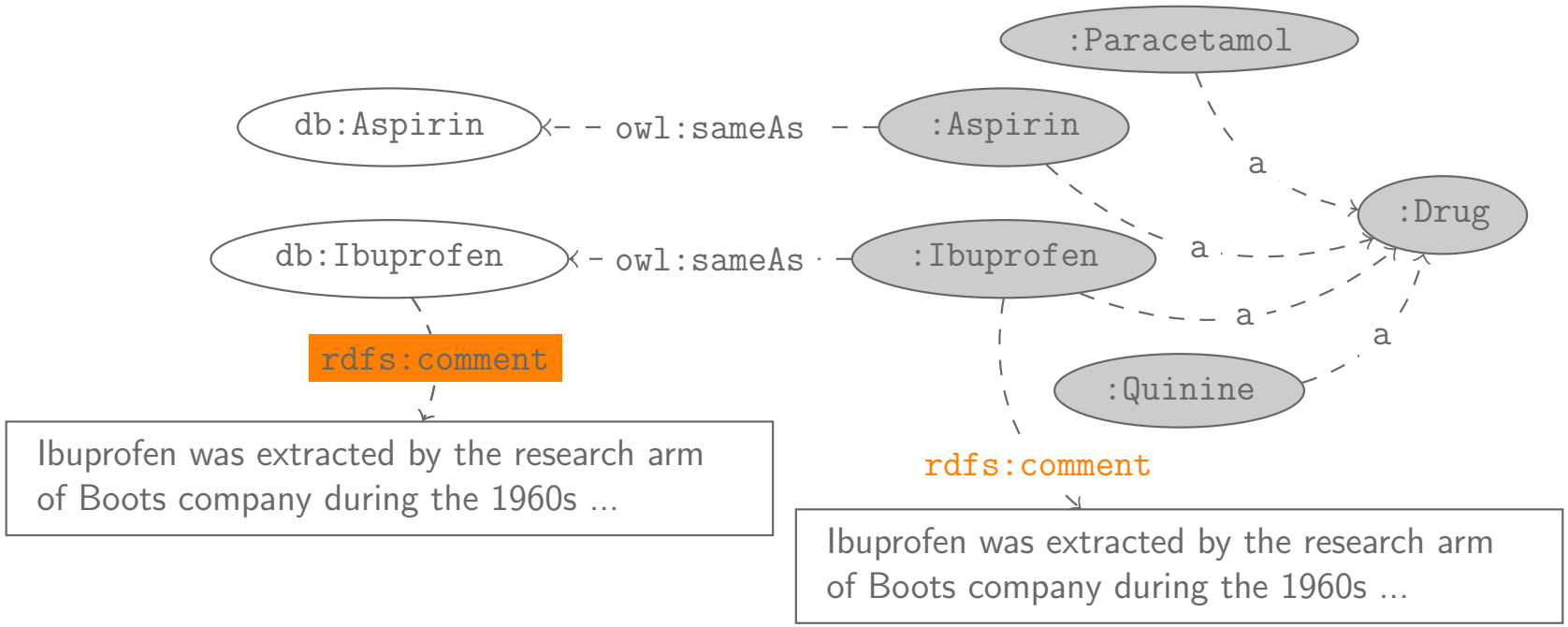
- Datasets are linked (e.g., using `owl:sameAs`)
- Deferences pre-specified set of predicates
- Adds found predicates to source the dataset



# Atomic Enrichment Functions

## I. Dereferencing atomic enrichment function

- Datasets are linked (e.g., using `owl:sameAs`)
- Deferences pre-specified set of predicates
- Adds found predicates to source the dataset



# Self-Configuration

## I. Dereferencing Enrichment Functions

- Finds the set of predicates $D_p$ from the enriched CBDs that are missing from source CBDs

### Non-enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen



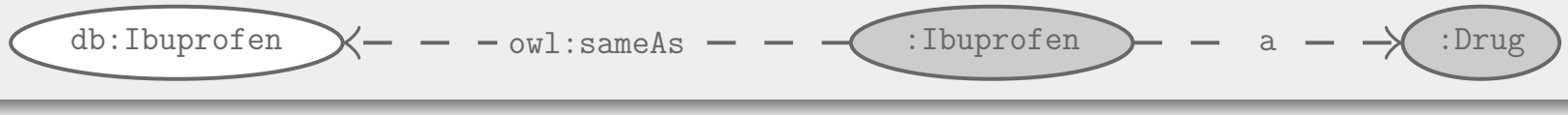

- $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

# Self-Configuration

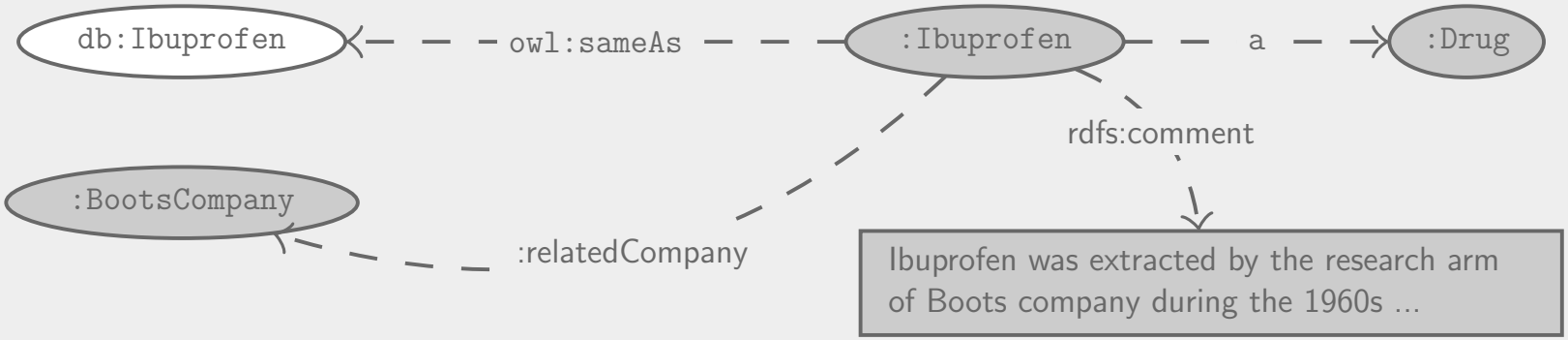
## I. Dereferencing Enrichment Functions

- Finds the set of predicates $D_p$ from the enriched CBDs that are missing from source CBDs

### Non-enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen



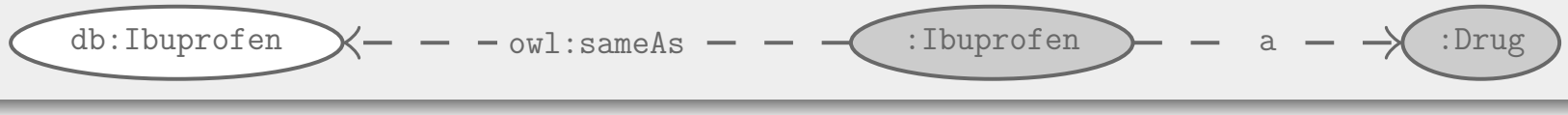

- $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

# Self-Configuration

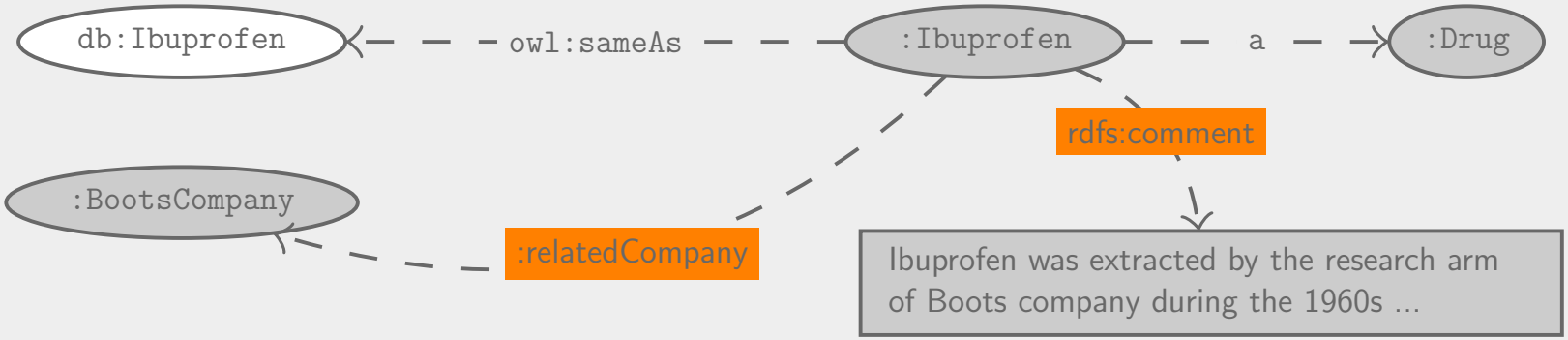
## I. Dereferencing Enrichment Functions

- Finds the set of predicates $D_p$ from the enriched CBDs that are missing from source CBDs

### Non-enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen



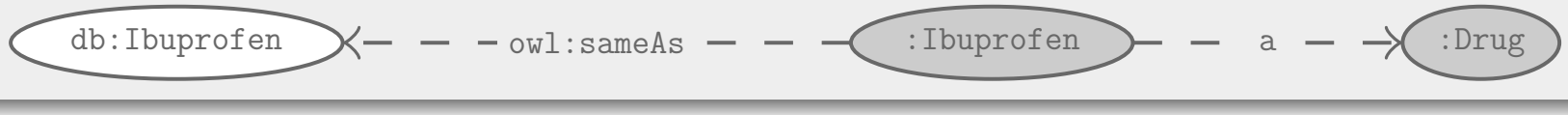

- $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

# Self-Configuration

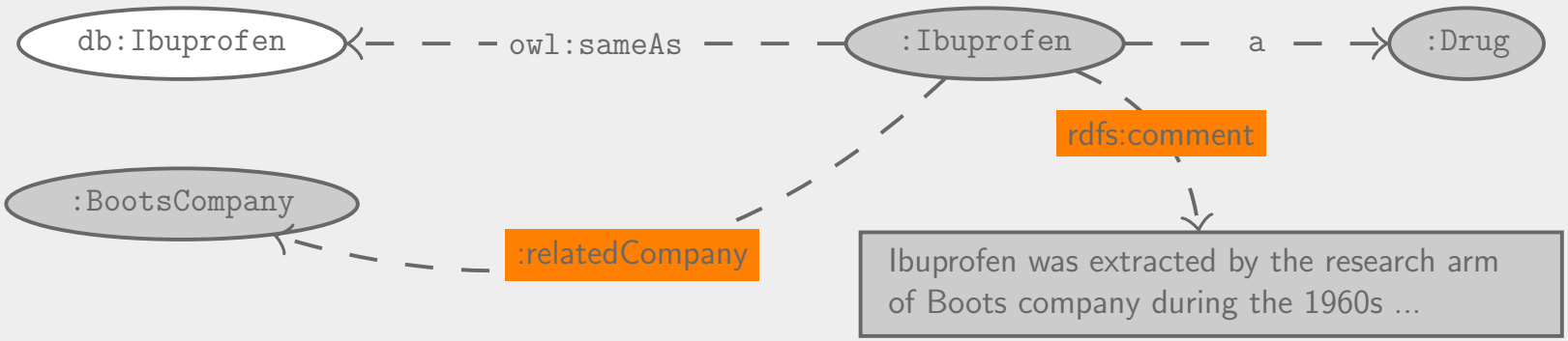
## I. Dereferencing Enrichment Functions

- Finds the set of predicates $D_p$ from the enriched CBDs that are missing from source CBDs

### Non-enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen




- $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

# Self-Configuration

## I. Dereferencing Enrichment Functions

- Dereferences $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

### CBD of Ibuprofen




- Finds only `rdfs:comment`, adds it to the source dataset

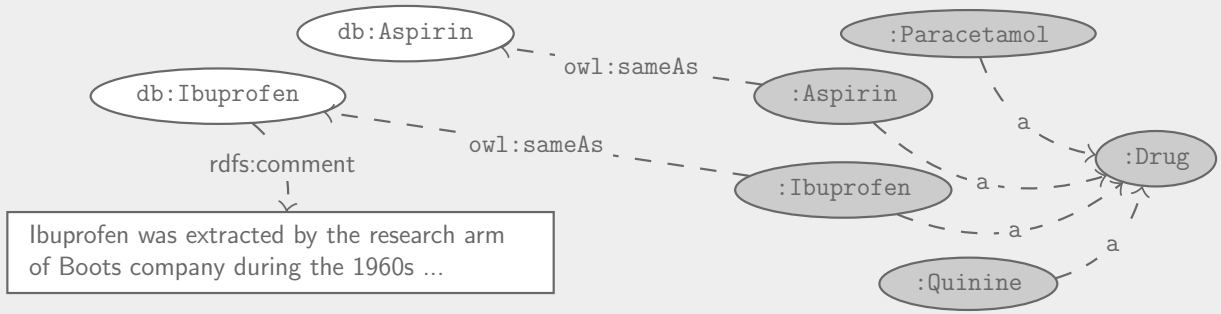
### Dereferencing enriched CBD of Ibuprofen

# Self-Configuration

## I. Dereferencing Enrichment Functions

- Dereferences $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

### CBD of Ibuprofen




- Finds only `rdfs:comment`, adds it to the source dataset

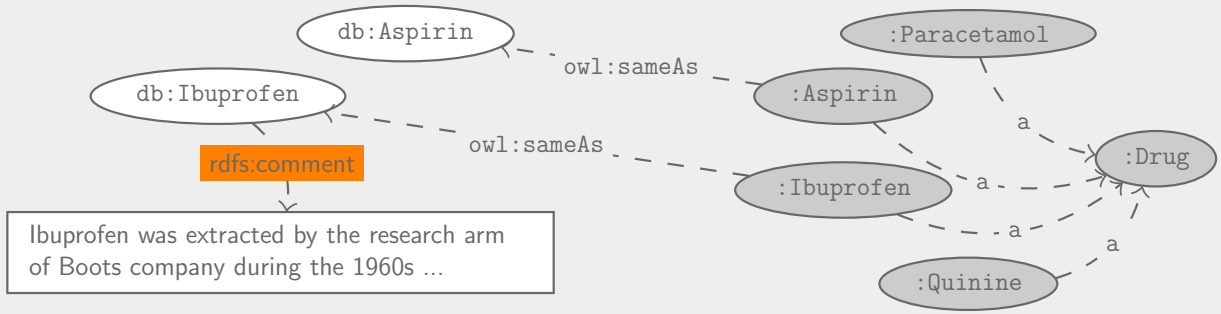
### Dereferencing enriched CBD of Ibuprofen

# Self-Configuration

## I. Dereferencing Enrichment Functions

- Dereferences $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

### CBD of Ibuprofen




- Finds only `rdfs:comment`, adds it to the source dataset

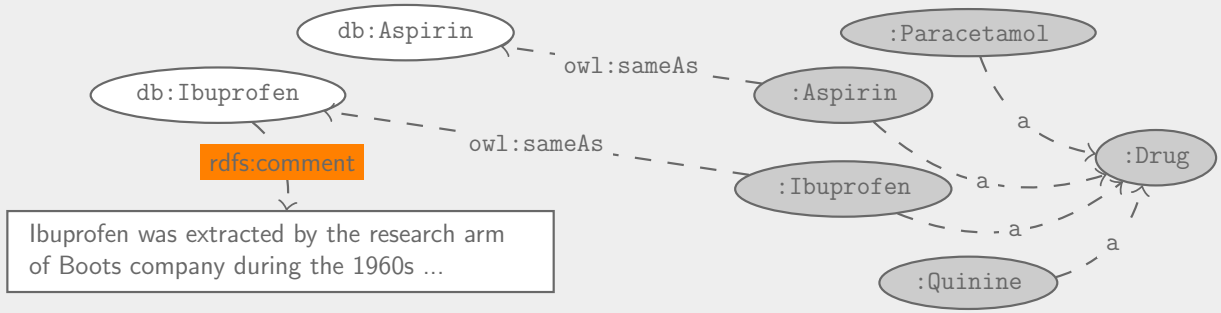
### Dereferencing enriched CBD of Ibuprofen

# Self-Configuration

## I. Dereferencing Enrichment Functions

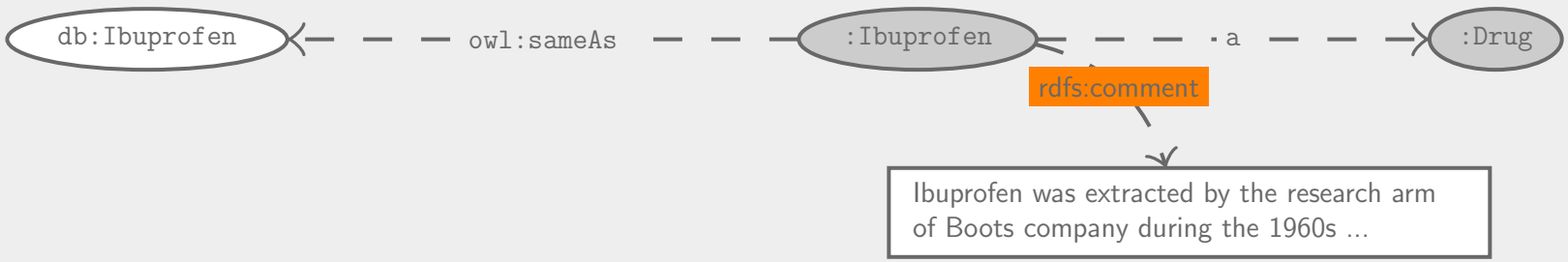
- Dereferences $D_p = \{\texttt{:relatedCompany}, \texttt{rdfs:comment}\}$

## CBD of Ibuprofen




- Finds only `rdfs:comment`, adds it to the source dataset

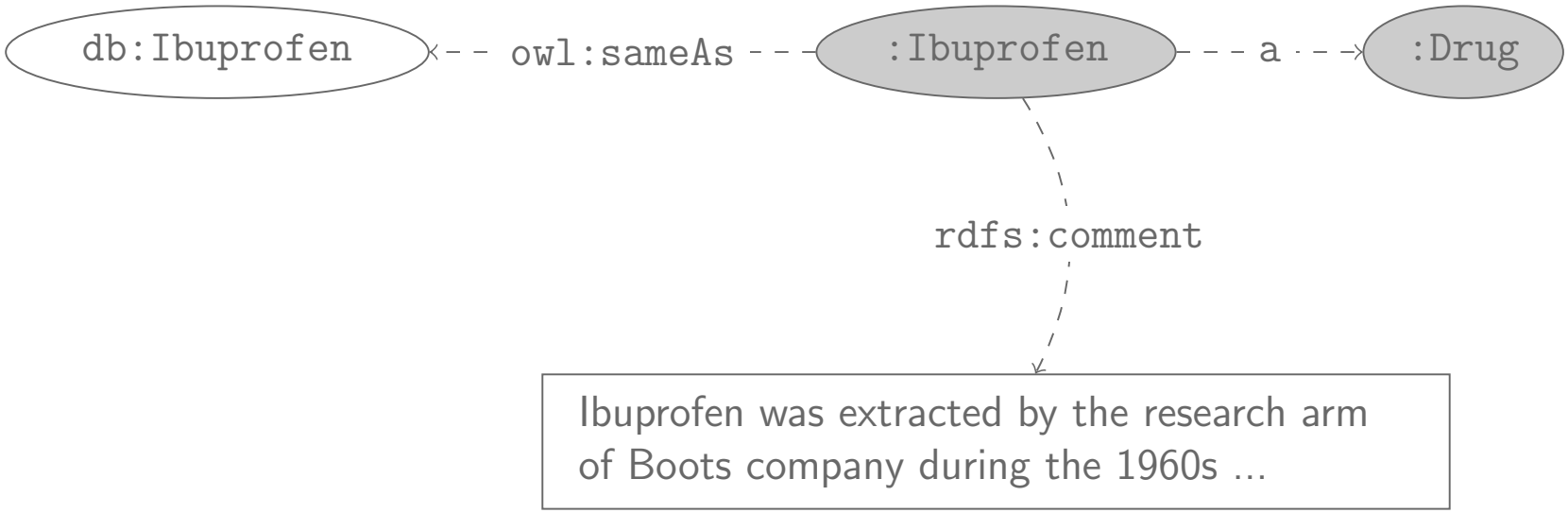
## Dereferencing enriched CBD of Ibuprofen



# Atomic Enrichment Functions

## II. NLP atomic enrichment function

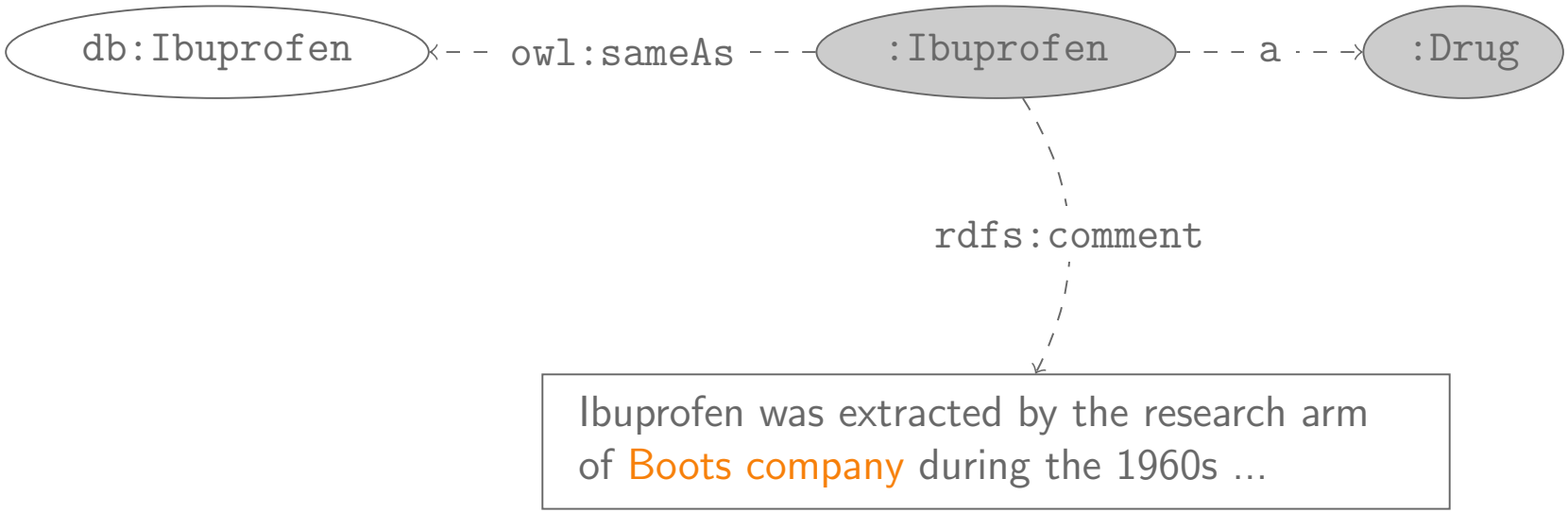
- Datatype objects contain unstructured information
- Uses *Named Entity Recognition* to extract implicit data
- Adds extracted entities to the source datasets



# Atomic Enrichment Functions

## II. NLP atomic enrichment function

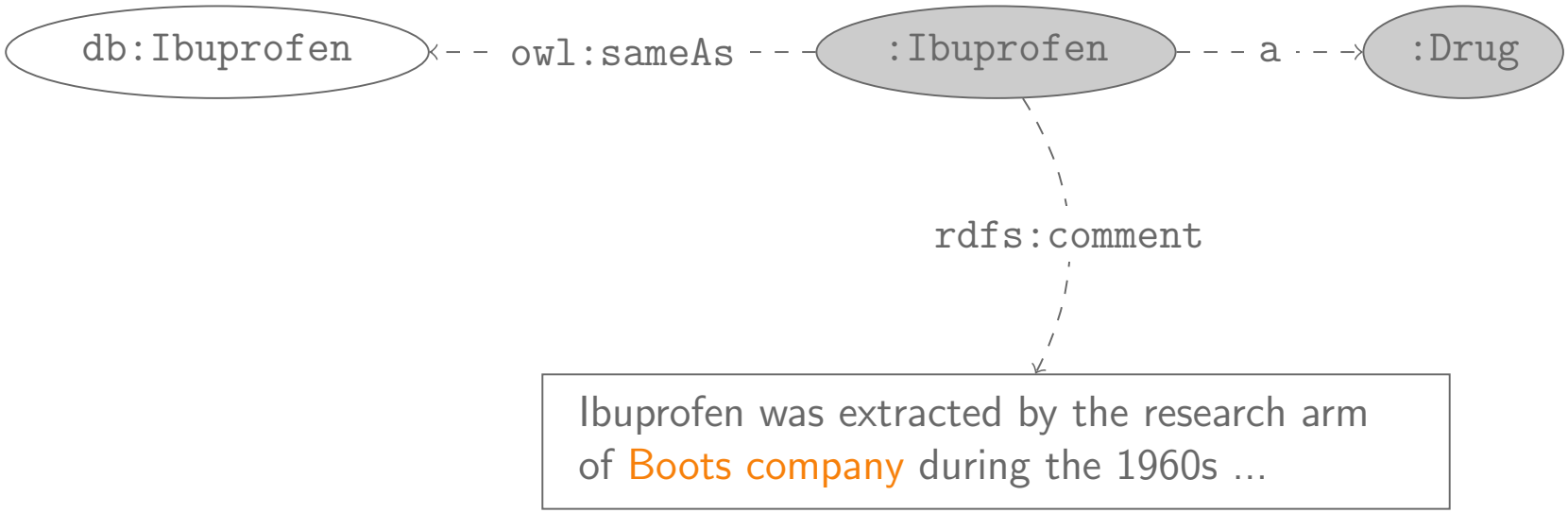
- Datatype objects contain unstructured information
- Uses *Named Entity Recognition* to extract implicit data
- Adds extracted entities to the source datasets



# Atomic Enrichment Functions

## II. NLP atomic enrichment function

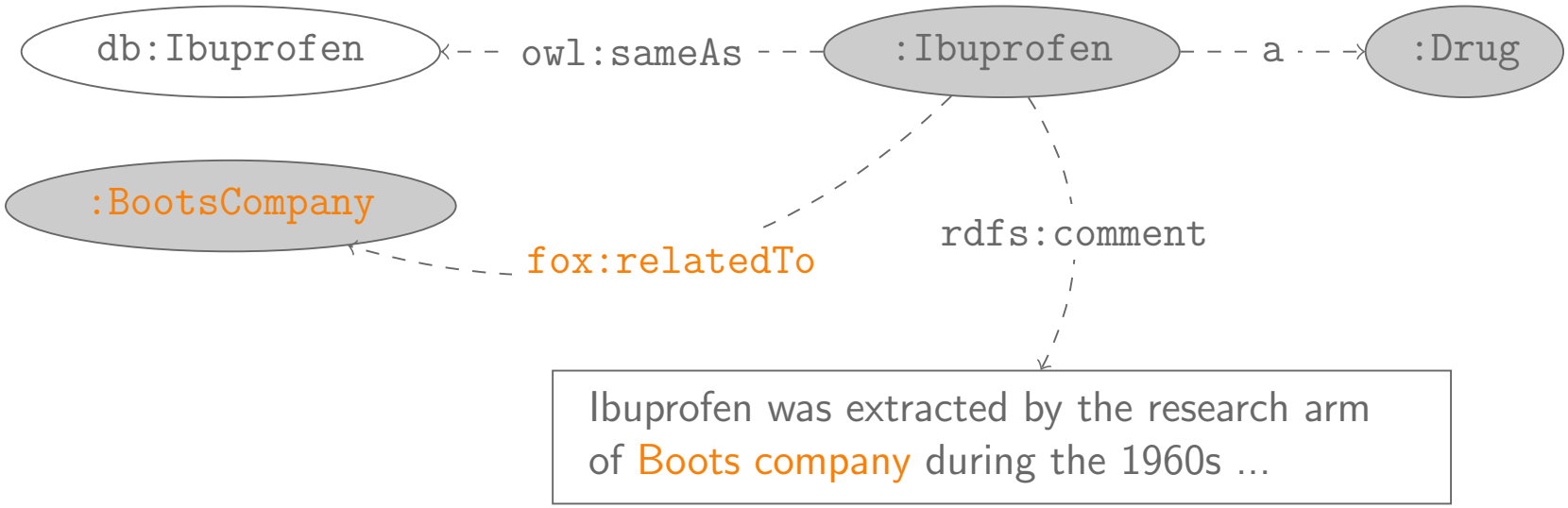
- Datatype objects contain unstructured information
- Uses *Named Entity Recognition* to extract implicit data
- Adds extracted entities to the source datasets



# Atomic Enrichment Functions

## II. NLP atomic enrichment function

- Datatype objects contain unstructured information
- Uses *Named Entity Recognition* to extract implicit data
- Adds extracted entities to the source datasets

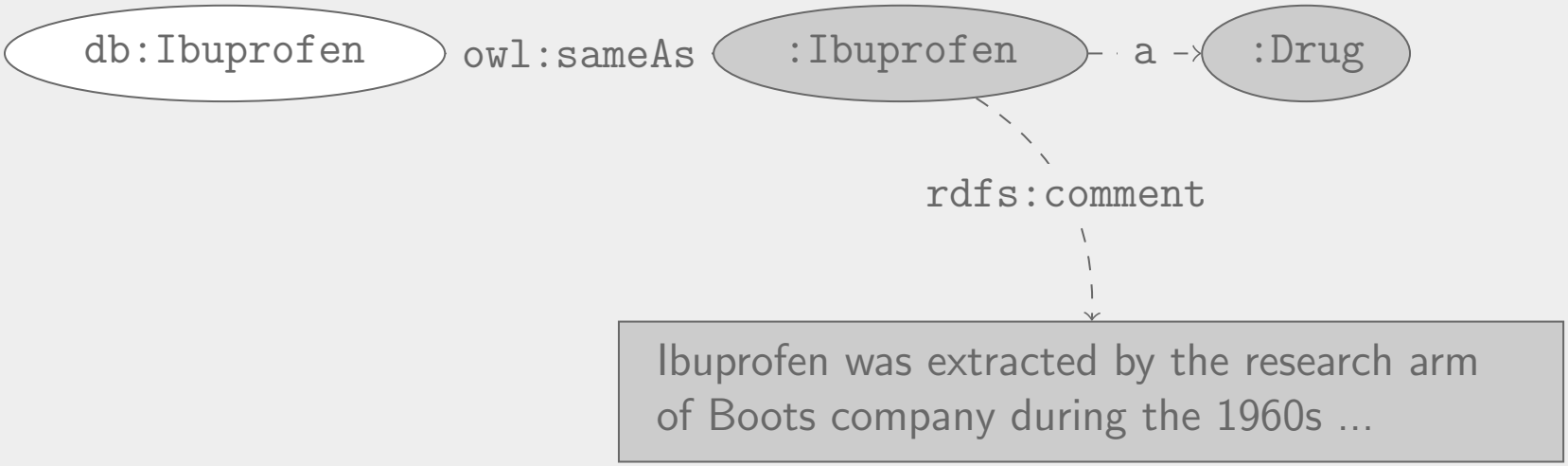


# Self-Configuration

## II. NLP Enrichment Function

- Extracts all possible named entity types
- Adds extracted entities to the source dataset

### NLP enriched CBD of Ibuprofen

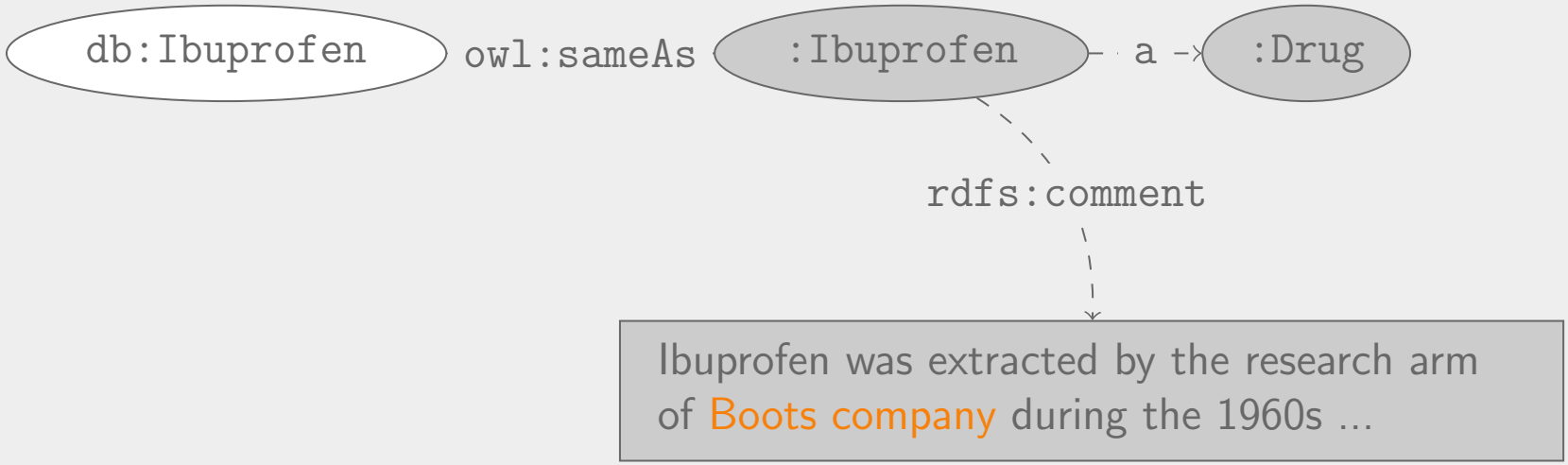


# Self-Configuration

## II. NLP Enrichment Function

- Extracts all possible named entity types
- Adds extracted entities to the source dataset

### NLP enriched CBD of Ibuprofen

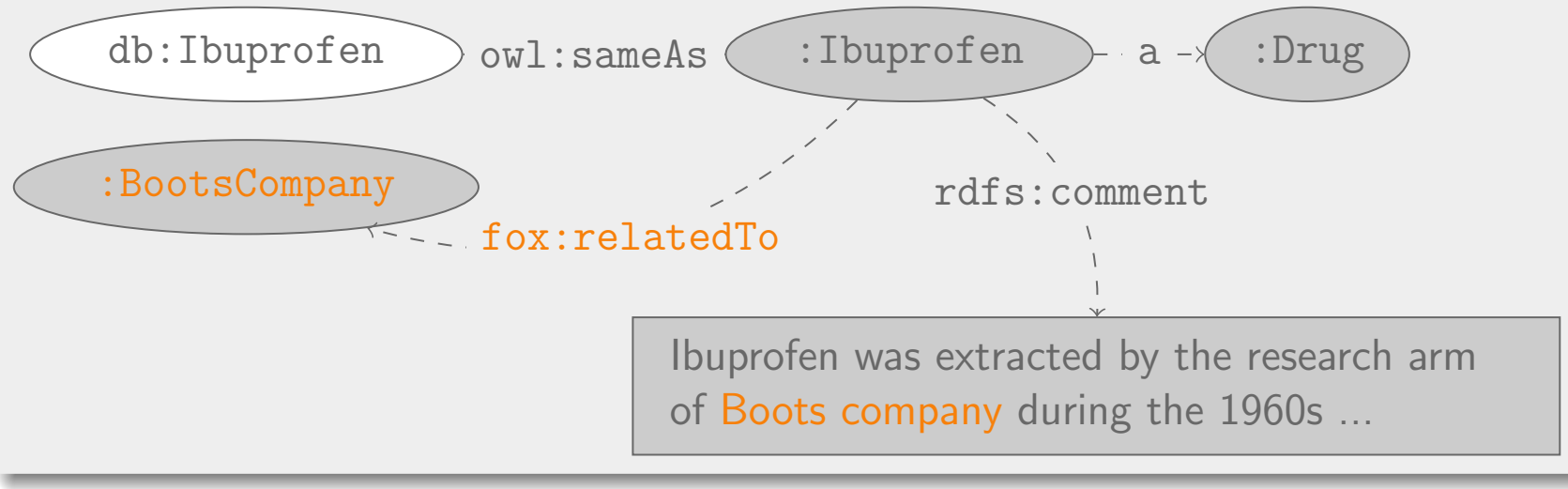


# Self-Configuration

## II. NLP Enrichment Function

- Extracts all possible named entity types
- Adds extracted entities to the source dataset

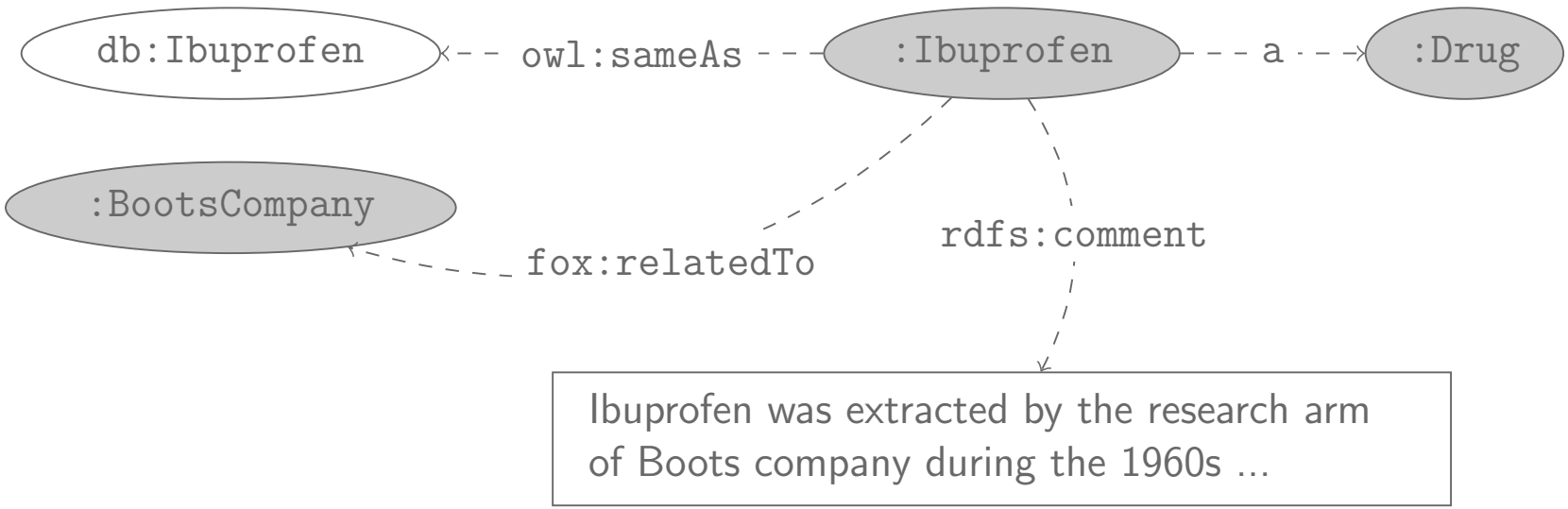
### NLP enriched CBD of Ibuprofen



# Atomic Enrichment Functions

## III. Predicate conformation atomic enrichment function

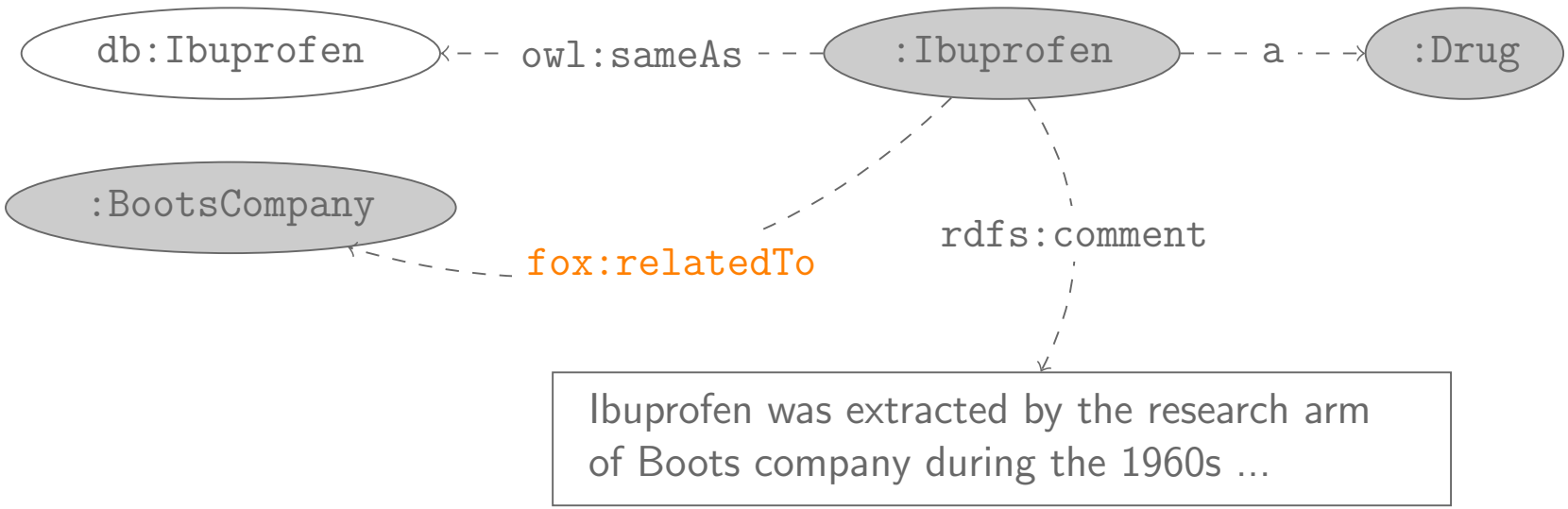
- Enriched datasets may contain diverse ontologies
- Predicate conformation maps a set of a pre-specified predicates to a target ontology



# Atomic Enrichment Functions

## III. Predicate conformation atomic enrichment function

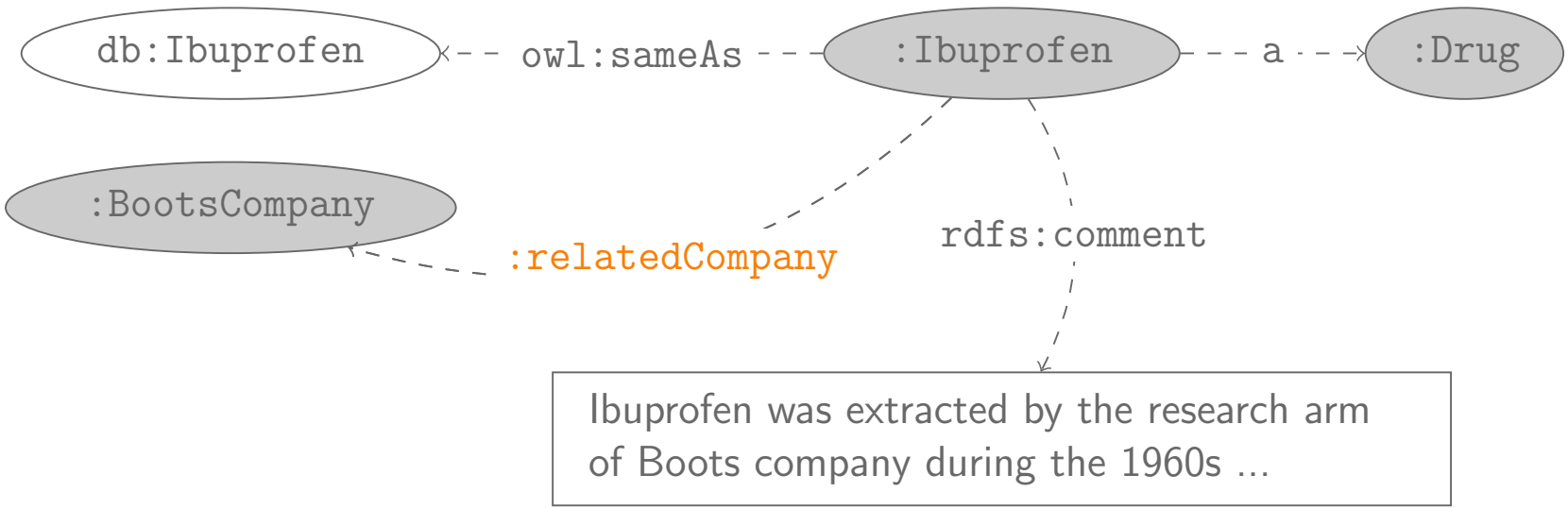
- Enriched datasets may contain diverse ontologies
- Predicate conformation maps a set of a pre-specified predicates to a target ontology



# Atomic Enrichment Functions

## III. Predicate conformation atomic enrichment function

- Enriched datasets may contain diverse ontologies
- Predicate conformation maps a set of a pre-specified predicates to a target ontology

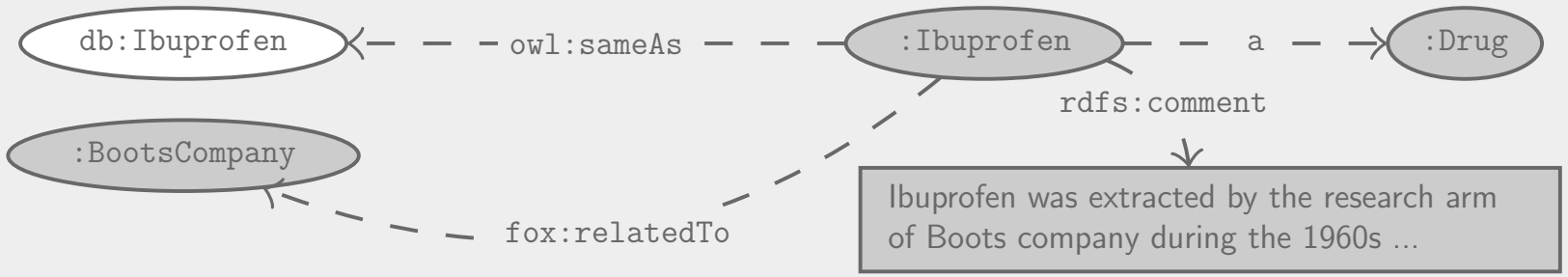


# Self-Configuration

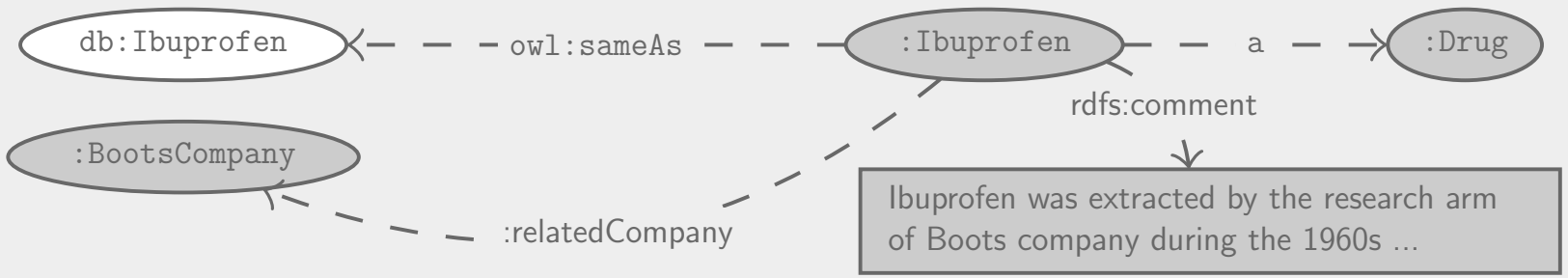
## III. Predicate conformation Enrichment Function

- Finds list of predicates $P_s$ and $P_t$ from the source resp. target datasets with the same subject and objects
- Changes each $P_s$ with its respective $P_t$

### NLP enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen (positive example target)

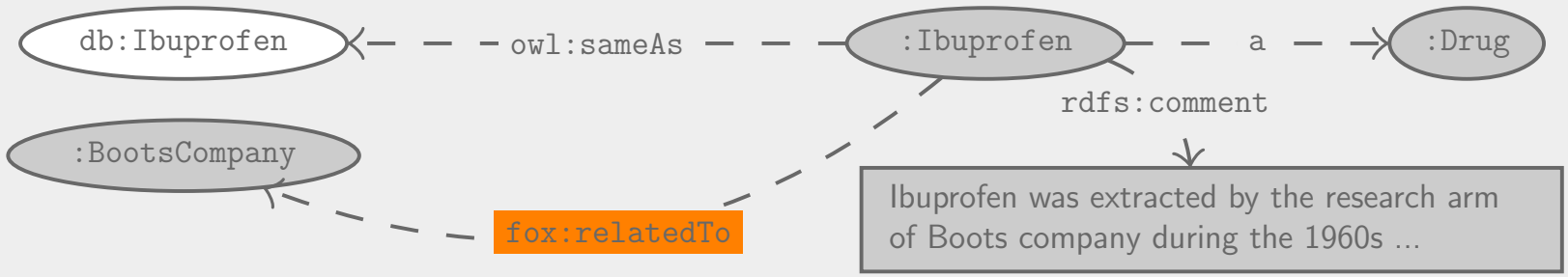


# Self-Configuration

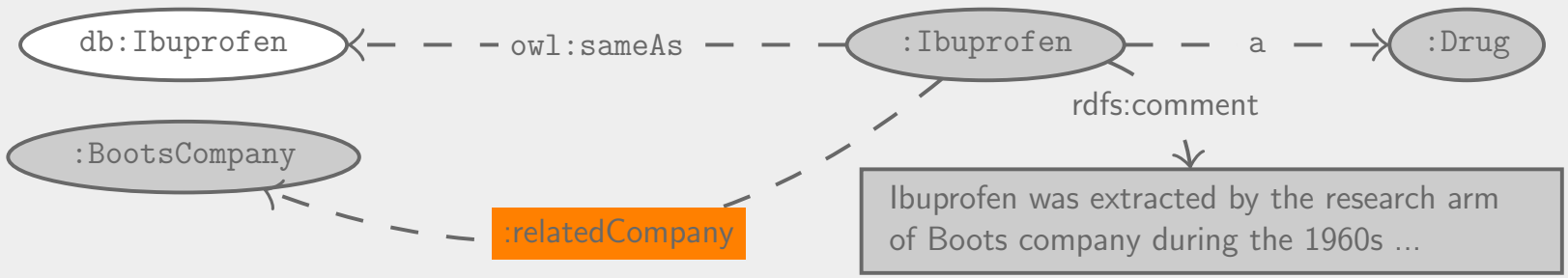
## III. Predicate conformation Enrichment Function

- Finds list of predicates $P_s$ and $P_t$ from the source resp. target datasets with the same subject and objects
- Changes each $P_s$ with its respective $P_t$

### NLP enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen (positive example target)

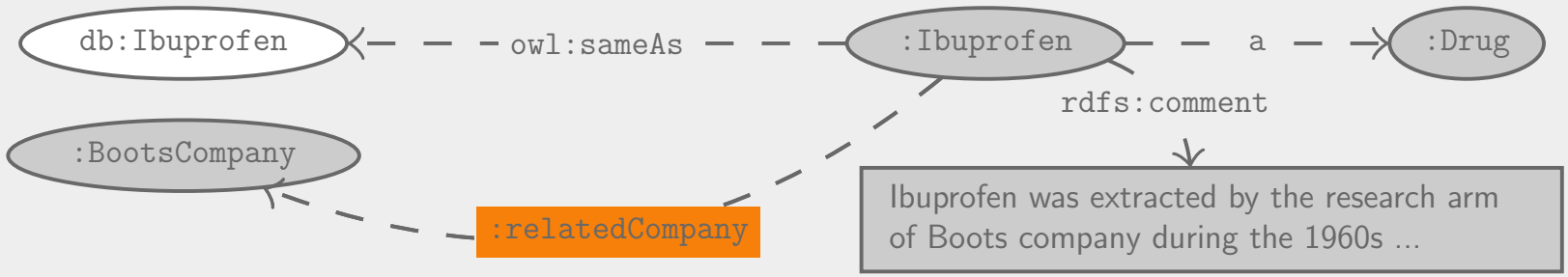


# Self-Configuration

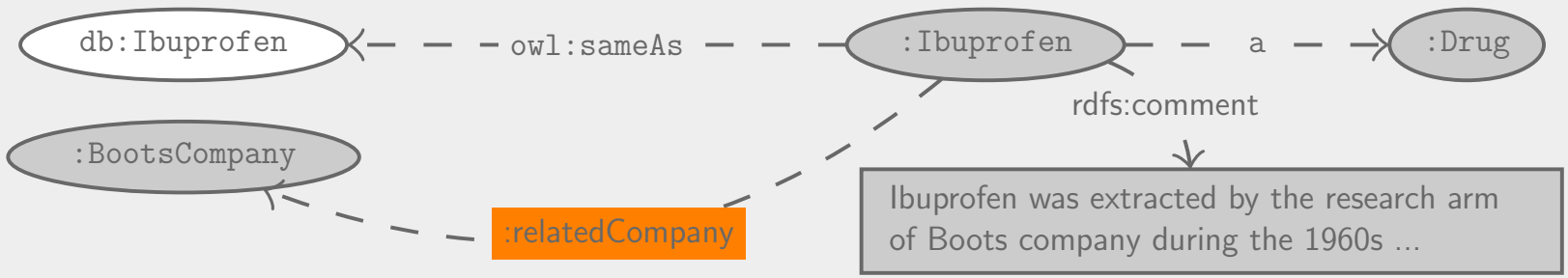
## III. Predicate conformation Enrichment Function

- Finds list of predicates $P_s$ and $P_t$ from the source resp. target datasets with the same subject and objects
- Changes each $P_s$ with its respective $P_t$

### NLP enriched CBD of Ibuprofen




### Enriched CBD of Ibuprofen (positive example target)



# KB Enrichment Refinement Operator

## Input

- Set of atomic enrichment functions $\mathcal{M}$
- Set of positive examples $\mathcal{E}$

## Refinement Operator

$$\rho(M) = \bigcup_{\forall m \in \mathcal{M}} M +\!\!+ m \qquad (+\!\!+ \text{ is the list append operator})$$

## Output

- Enrichment pipeline $M$

# KB Enrichment Refinement Operator

## Input

- Set of atomic enrichment functions $\mathcal{M}$
- Set of positive examples $\mathcal{E}$

## Refinement Operator

$$\rho(M) = \bigcup_{\forall m \in \mathcal{M}} M +\!\!+ \ m \qquad (+\!\!+ \text{ is the list append operator})$$

## Output

- Enrichment pipeline $M$

# KB Enrichment Refinement Operator

## Input

- Set of atomic enrichment functions $\mathcal{M}$
- Set of positive examples $\mathcal{E}$

## Refinement Operator

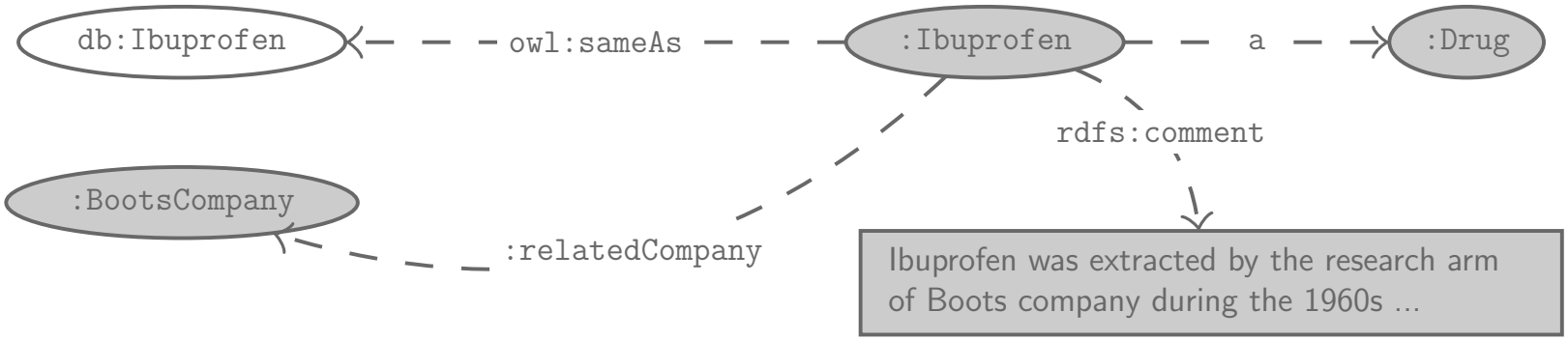
$$\rho(M) = \bigcup_{\forall m \in \mathcal{M}} M +\!\!+ \ m \qquad (+\!\!+ \text{ is the list append operator})$$

## Output

- Enrichment pipeline $M$

# Positive Example


Non-enriched CBD of Ibuprofen



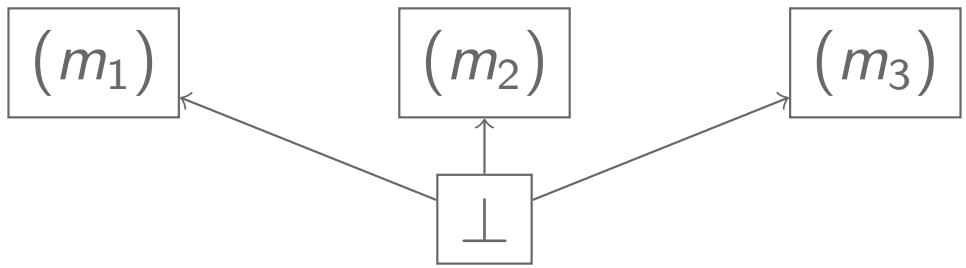

Enriched CBD of Ibuprofen

# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node

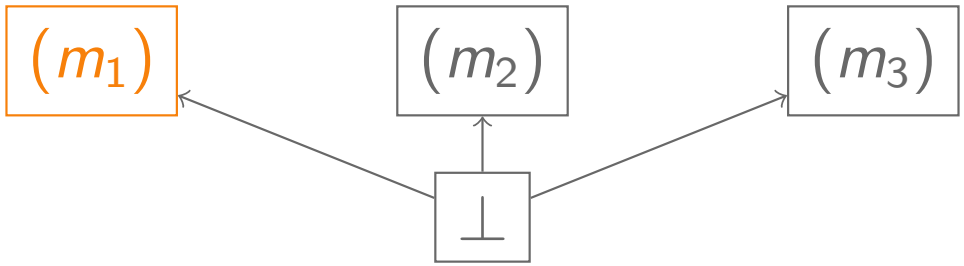
# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



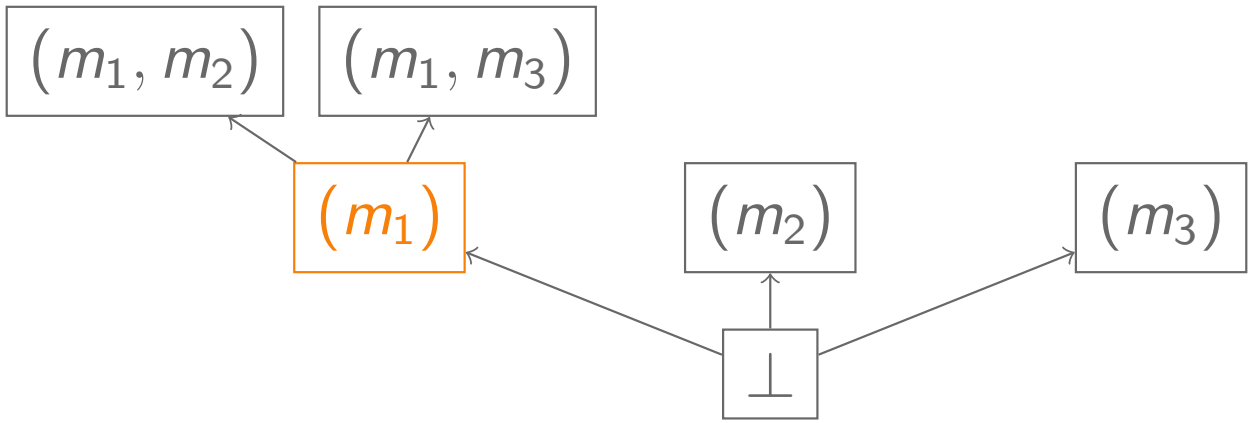
# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



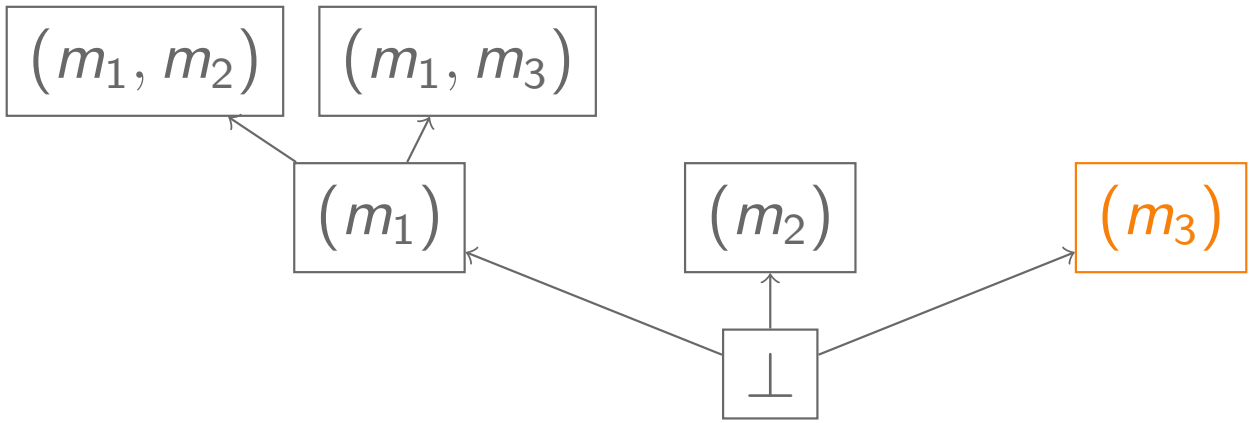
# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



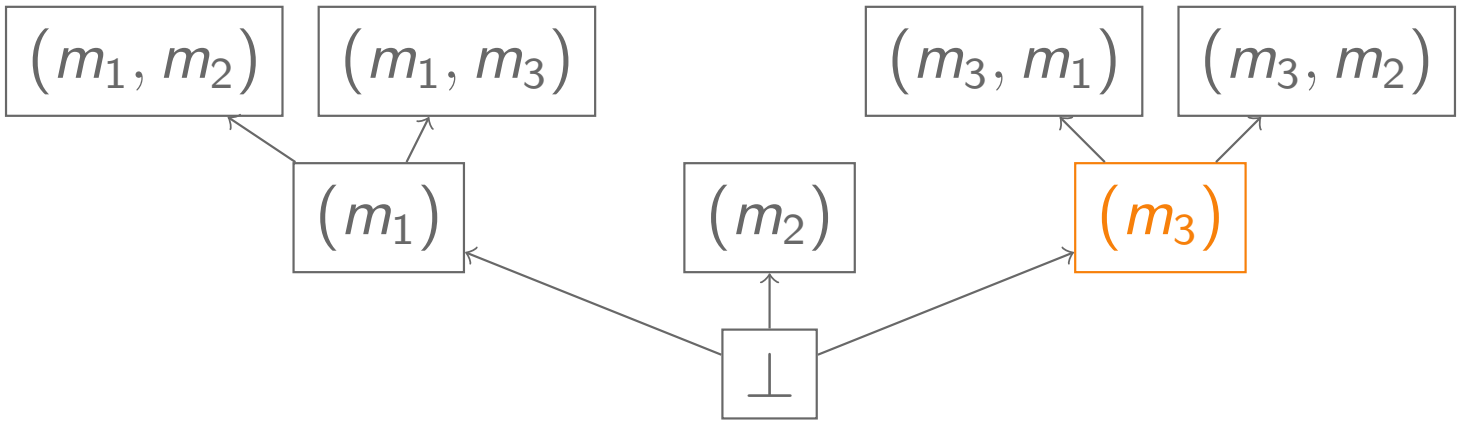
# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



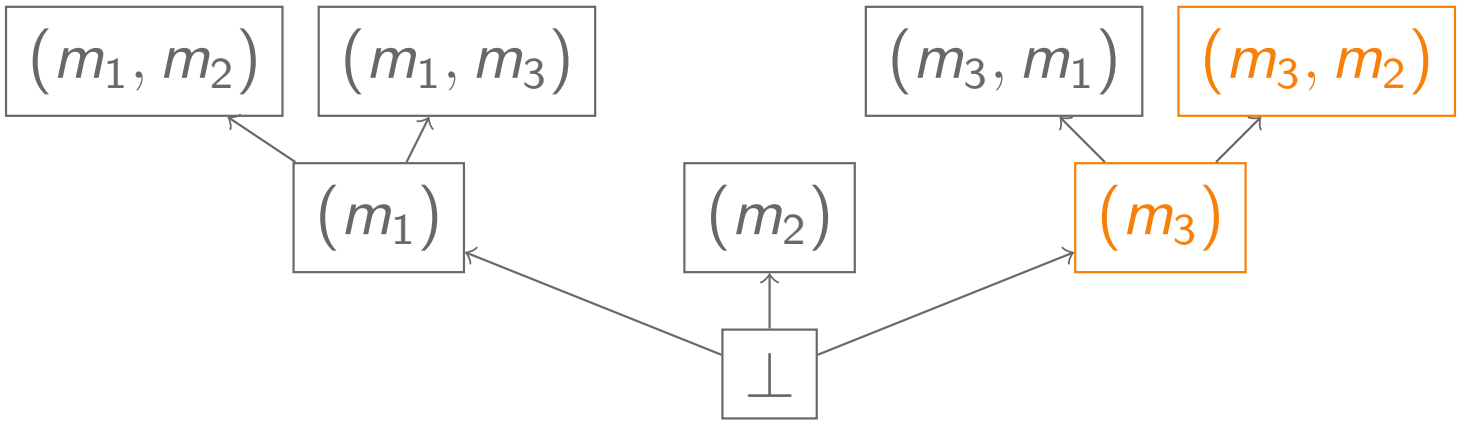
# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



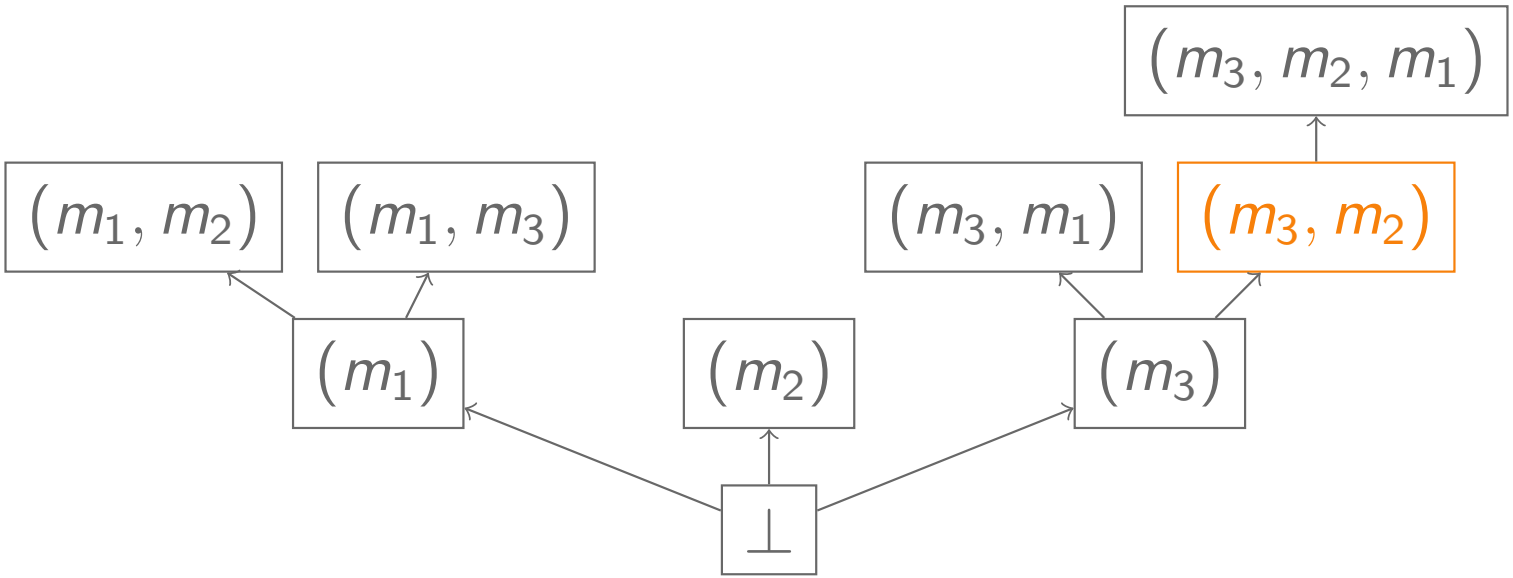
# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



# Learning Algorithm

1. Start by empty enrichment pipeline $M = \perp$
2. Self-configure all $m_i \in \mathcal{M}$, add as child to $\perp$
3. Select most promising node
4. Expand most promising node



# Most Promising Node Selection

## Node complexity $c(n)$

- Linear combination of the node's children count and level

## Node fitness $f(n)$

- Difference between node's enrichment pipeline F-measure and weighted complexity, $f(n) = F(n) - \omega.c(n)$
- $\omega$ controls the tradeoff between
  - Greedy search ($\omega = 0$)
  - Search strategies closer to breadth-first search ($\omega > 0$).

## Most promising node

- The leaf node with the maximum fitness through the whole refinement tree

# Most Promising Node Selection

## Node complexity $c(n)$

- Linear combination of the node's children count and level

## Node fitness $f(n)$

- Difference between node's enrichment pipeline F-measure and weighted complexity, $f(n) = F(n) - \omega.c(n)$
- $\omega$ controls the tradeoff between
  - Greedy search ($\omega = 0$)
  - Search strategies closer to breadth-first search ($\omega > 0$).

## Most promising node

- The leaf node with the maximum fitness through the whole refinement tree

# Most Promising Node Selection

## Node complexity $c(n)$

- Linear combination of the node's children count and level

## Node fitness $f(n)$

- Difference between node's enrichment pipeline F-measure and weighted complexity, $f(n) = F(n) - \omega.c(n)$
- $\omega$ controls the tradeoff between
  - Greedy search ($\omega = 0$)
  - Search strategies closer to breadth-first search ($\omega > 0$).

## Most promising node

- The leaf node with the maximum fitness through the whole refinement tree

# Outline

# Experimental Setup

## Datasets

- 1 manual experimental enrichment pipelines for *Jamendo*
- 2 manual experimental enrichment pipelines for *DrugBank*
- 5 manual experimental enrichment pipelines for *DBpedia* (`AdministrativeRegion`)

## Learning Algorithm

- 6 atomic enrichment functions
- Termination criterion:
  - Maximum number of iterations of 10
  - Optimal enrichment pipeline found (F-score = 1)

# Experimental Setup

## Datasets

- 1 manual experimental enrichment pipelines for *Jamendo*
- 2 manual experimental enrichment pipelines for *DrugBank*
- 5 manual experimental enrichment pipelines for *DBpedia* (`AdministrativeRegion`)

## Learning Algorithm

- 6 atomic enrichment functions
- Termination criterion:
  - Maximum number of iterations of 10
  - Optimal enrichment pipeline found (F-score = 1)

# Configuration of the Search Strategy

- Node fitness
  $f(n) = F(n) - \omega.c(n)$
- $\omega$ controls the tradeoff between
  - Greedy search ($\omega = 0$)
  - Search strategies closer to breadth first search ($\omega > 0$).
- Result: $\omega = 0.75$ leads to the best results

| $\omega$ | $P$ | $R$ | $F$ |
|---|---|---|---|
| 0 | 1.0 | 0.99 | 0.99 |
| 0.25 | 1.0 | 0.99 | 0.99 |
| 0.50 | 1.0 | 0.99 | 0.99 |
| 0.75 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 0.99 | 0.99 |

# Effect of Positive Examples

| Manual $M$ | Examples count | Size of $M$ | Time $M(KB)$ | Size of learned $M'$ | Time $M'(KB)$ | Learn Time | Iterations count | $F$-score |
|---|---|---|---|---|---|---|---|---|
| $M^1_{DBpedia}$ | 1 | 1 | 0.2 | 1 | 1.6 | 1.3 | 1 | 1.0 |
| | 2 | 1 | 0.2 | 1 | 1.8 | 1.3 | 1 | 1.0 |
| $M^2_{DBpedia}$ | 1 | 2 | 23.3 | 1 | 0.1 | 0.2 | 1 | 0.99 |
| | 2 | 2 | 15 | 2 | 17 | 0.3 | 9 | 0.99 |
| $M^3_{DBpedia}$ | 1 | 3 | 14.7 | 3 | 15.2 | 6.1 | 9 | 0.99 |
| | 2 | 3 | 15 | 2 | 15.1 | 0.1 | 9 | 0.99 |
| $M^4_{DBpedia}$ | 1 | 4 | 0.4 | 2 | 0.1 | 0.7 | 2 | 0.99 |
| | 2 | 4 | 0.6 | 2 | 0.3 | 0.9 | 2 | 0.99 |
| $M^5_{DBpedia}$ | 1 | 5 | 22 | 2 | 0.1 | 0.7 | 2 | 1.0 |
| | 2 | 5 | 25.5 | 2 | 0.2 | 0.9 | 2 | 1.0 |
| $M^1_{DrugBank}$ | 1 | 2 | 3.5 | 1 | 4.1 | 0.1 | 10 | 0.99 |
| | 2 | 2 | 3.6 | 1 | 3.4 | 0.1 | 10 | 0.99 |
| $M^2_{DrugBank}$ | 1 | 3 | 25.2 | 1 | 0.1 | 0.1 | 10 | 0.99 |
| | 2 | 3 | 22.8 | 1 | 0.1 | 0.1 | 61 | 0.99 |
| $M^1_{Jamendo}$ | 1 | 1 | 10.9 | 2 | 10.6 | 0.1 | 2 | 0.99 |
| | 2 | 1 | 10.4 | 2 | 10.4 | 0.1 | 1 | 0.99 |

# Outline

# Conclusion and Future Work

## Conclusion

- Presented self-configuring atomic enrichment functions
- Presented an approach for learning enrichment pipelines based on a refinement operator
- Showed that our approach can easily reconstruct manually created enrichment pipelines

## Future Work

- Parallelize the algorithm on several CPUs as well as load balancing
- Support directed acyclic graphs as enrichment specifications by allowing to split and merge datasets
- Pro-active enrichment strategies and active learning

# Conclusion and Future Work

## Conclusion

- Presented self-configuring atomic enrichment functions
- Presented an approach for learning enrichment pipelines based on a refinement operator
- Showed that our approach can easily reconstruct manually created enrichment pipelines

## Future Work

- Parallelize the algorithm on several CPUs as well as load balancing
- Support directed acyclic graphs as enrichment specifications by allowing to split and merge datasets
- Pro-active enrichment strategies and active learning


# Thank You!

# Questions?

Mohamed Sherif
Augustusplatz 10
D-04109 Leipzig
sherif@informatik.uni-leipzig.de
http://aksw.org/MohamedSherif
http://aksw.org/Projects/DEER

#akswgroup