Complex Schema Mapping and Linking Data: Beyond Binary Predicates

Jacobo Rouces
Aalborg University
jacobo@rouces.org

Gerard de Melo
Tsinghua University
gdm@demelo.org

Katja Hose
Aalborg University
khose@cs.aau.dk
Overview

- Data Heterogeneity in the LOD Cloud
- FrameBase
- Creation of complex mappings with FrameBase hub
- Conclusion & Future Work
Data Heterogeneity in the LOD Cloud

Using Direct Binary Relations *(used as “default” mode in most KBs)*
Data Heterogeneity in the LOD Cloud

RDF reification (YAGO)
Data Heterogeneity in the LOD Cloud

Using “eventive” subproperties (Nguyen et al, WWW 2014)
Data Heterogeneity in the LOD Cloud

Ad-hoc and few...

Neo-davidsonian representations (used to an extent in most KBs that include events. E.g. Freebase)
Linking What?
Linking What?

John

same as

Mary

same as

same as

if(f)

Date

1964

wasMarriedAtDate

1964

wasMarriedWith

Mary

John

John

Mary

wasMarriedAtDate
Linking What?
Linking What?

- Linking data is not linking entities
- Current efforts focus mostly on linking entities one to one
FrameBase: schema

- Core: RDFS schema to represent knowledge using neo-Davidsonian approach with a wide and extensible vocabulary of
  - **Frames.** In a hierarchy. Frames are “events, situations, eventualities…”
  - **Frame Elements.** Outgoing properties representing frame-specific semantic roles
FrameBase

*schema*

- Vocabulary based on NLP resources (FrameNet+WordNet)
  - This provides connection with natural language and semantic role labeling systems. It clusters near-equivalents.
FrameBase

schema

![Diagram of FrameBase schema](image-url)
FrameBase

schema

John

isMarriedWith

isMarriedAtDate

Partner 1

Wedding.n
Marry.v
Marriage.n

Time

1964

Partner 2

isMarriedAtDate

Mary
FrameBase: ReDer rules

- Two-layered structure:
  - Create two levels of reification, and Reification-Dereification (ReDer) inference rules (horn clauses) that connect them.
  - Reified knowledge using frames and frame elements
  - Dereified knowledge using direct binary predicates (DBPs)
- Currently ~15000 rules/DBPs

```prolog
?f a :frame-Separating-partition.v AND
?f :fe-Separating-Whole ?s AND
?f :fe-Separating-Parts ?o IFF
?s ..-isPartitionedIntoParts ?o
```
FrameBase

Example

yago:Nobel_Prize

yago:A_Einstein

fe-Win_prize-competition

fe-Win_prize-prize

...-competitor

...-time

1921^xsd:date
FrameBase

Example

yago:Nobel_Prize

fe-Win_prize-competition
fe-Win_prize-prize

yago:A_Einstein

fe-Working_on-agent

1905^xsd:date

1921^xsd:date

yago:Photoelectric_effect

frame:Working_on-work.n

...-domain

...-explanation

...-time

...-competitor

...-time

...-time
Example

FrameBase

yago:Nobel_Prize

fe-Win_prize-competition
fe-Win_prize-prize

yago:A_Einstein

winsBy Competitor

:frame-Win_prize-win.v

isWonAtTime

1921^xsd:date

yago:Photoelectric_effect

frame:Working_on-work.n

worksAtTime

1905^xsd:date
Creation of complex mappings

- Complex mappings between FrameBase and external KBs. Built in three steps:

1. Creating ReDer rules and DBPs in FrameBase

2. Canonicalizing predicate names from external Kbs

3. Matching DBPs with external predicate names

\[
?f \ a :frame\text{-}\text{Separating\text{-}partition}.v \\
\text{AND} \ ?f \ :fe\text{-}\text{Separating\text{-}Whole} \ ?s \\
\text{AND} \ ?f \ :fe\text{-}\text{Separating\text{-}Parts} \ ?o \\
\text{IFF} \ ?s \ ..\text{isPartitionedIntoParts} \ ?o
\]

\[
\text{somekb:splitInto} \\
\rightarrow \\
\text{somekb:isSplitInto}
\]

\[
\text{sim('} \text{is split into'}, \\
\text{'is partitioned into parts')}
\]

\[
?f \ a :frame\text{-}\text{Separating\text{-}partition}.v \\
\text{AND} \ ?f \ :fe\text{-}\text{Separating\text{-}Whole} \ ?s \\
\text{AND} \ ?f \ :fe\text{-}\text{Separating\text{-}Parts} \ ?o \\
\text{IFF} \ ?s \ \text{somekb:splitInto} \ ?o
\]
Creation of complex mappings

Step 1. Creating DBPs in FrameBase

- In [1], DBPs are created with verbs and nouns as heads. We extend the approach to deal with adjectives as well.
- We use syntactic annotations from FrameNet

\[\begin{array}{|c|}
\hline
\text{Creation Rule: Copula+Adjective} \\
\hline
\text{Create DBP with name "is LU PREP FE-O" if} \\
\text{ISADJECTIVE(LU) AND phrase-type-o==PP[PREP]} \\
\text{AND grammatical-function-s==Ext} \\
\text{AND grammatical-function-o==Dep} \\
\hline
\end{array}\]

\[
\text{?s dbp-Sound\_level-is\_Loud\_To\_Degree \ ?o} \\
\text{f type frame-Sound\_level-loud.a} \\
\text{f fe-Sound\_level-Entity \ ?s} \\
\text{f fe-Sound\_level-Degree \ ?o}
\]

Creation of complex mappings

Step 1. Creating DBPs in FrameBase

- In [1], DBPs are created with verbs and nouns as heads. We extend the approach to deal with adjectives as well.
- We use syntactic annotations from FrameNet

<table>
<thead>
<tr>
<th>Creation Rule: Copula+Adjective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create DBP with name “is LU PREP FE-o” if</td>
</tr>
<tr>
<td>ISADJECTIVE(LU) AND phrase-type-o==PP[PREP]</td>
</tr>
<tr>
<td>AND grammatical-function-s==Ext</td>
</tr>
<tr>
<td>AND grammatical-function-o==Dep</td>
</tr>
</tbody>
</table>

\[
?s \text{ dbp-Sound_level-becomesLoudToDegree} \quad \text{and} \quad \text{dbp-Sound_level-seemsLoudToDegree} \quad ?o
\]

\[
\downarrow
\]

f type frame-Sound_level-loud.a
f fe-Sound_level-Entity ?s
f fe-Sound_level-Degree ?o
f’ type frame-Becoming-become.v
f’ fe-Becoming-Entity ?s
f’ fe-Becoming-Final_state f

\[
\downarrow
\]

f type frame-Sound_level-loud.a
f fe-Sound_level-Entity ?s
f fe-Sound_level-Degree ?o
f’ type frame-Appearance-seem.v
f’ fe-Appearance-Phenomenon ?s
f’ fe-Appearance-Inference f
Creation of complex mappings

Step 2. Canonicalizing predicate names from external Kbs

Apply a set of rules for name transformations:

- If the name $p$ of a property is a past participle, it can be extended with the prefix “is” (without postfix “of”). Ex: “created” → “is created”

- If the name $p$ of a property is a noun or a noun phrase, and a range is declared for the property, let $X$ be a set containing $p$’s name and the hypernyms of all its word senses (obtained from WordNet). If for any element $x$ in $X$, $p$ is a substring of $x$ or $x$ is a substring of $p$, then $p$ can be extended with the prefix “has”. Ex: “creator” with range ”person” → “has creator”

- The same rule as above, but using the domain instead of the range, which allows $p$ to be extended with the prefix “is” and postfix “of”. Ex: “creator” with domain “person” → “is creator of”
Creation of complex mappings

Step 2. Canonicalizing predicate names from external Kbs

- If the property is symmetric, we can introduce extensions both with “has” and with “is”+ . . . +“of”. Ex: “sibling” is owl:symmetric → “has sibling”, “is sibling of”;

- For every property p corresponding to the pattern “is X of”, an inverse property can be created of the form “has X”. Ex: “is mentor of” → ^“has mentor”

- For every property p corresponding to the pattern “has X”, an inverse property can be created of the form “is X of”. Ex: “has mentor” → ^“is mentor of”
Creation of complex mappings

Step 3. Matching DBPs with external predicate names

For each canonicalized Source Dataset Property (SDP), maximize over all DBPs:

\[
0.7 \cdot \text{bag of words labels} + 0.1 \cdot \text{bag of words domain, range, etc} + 0.1 \cdot \text{"core" properties in reified pattern} + 0.1 \cdot \text{0.1's for resolving ties} + 0.1 \cdot \text{Difficult to use supervised ML: very low IA agreement for gold standards}
\]
Creation of complex mappings

Results

- Canonicalized properties from source KB (DBpedia)
  - Examples:

<table>
<thead>
<tr>
<th>Source property</th>
<th>Canonicalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>currently run by</td>
<td>is currently run by</td>
</tr>
<tr>
<td>golden raspberry award</td>
<td>has golden raspberry award</td>
</tr>
<tr>
<td>statistic</td>
<td>is statistic of</td>
</tr>
<tr>
<td>link title</td>
<td>has link title</td>
</tr>
<tr>
<td>first leader</td>
<td>has first leader</td>
</tr>
</tbody>
</table>

Precision: 85%
Creation of complex mappings

**Results**

- **Integration rules (DBpedia)**

  - **Examples:**

  ```rdf
  CONSTRUCT {
  _:r a :frame-Education_teaching-school.v 
  _:r :fe-Education_teaching-Student ?S .  
  _:r :fe-Education_teaching-Skill ?O .  
  } WHERE {
  }
  
  CONSTRUCT {
  _:r a :frame-Appearance-smell.v 
  _:r :fe-Appearance-Phenomenon ?S .  
  } WHERE {
  }
  
  CONSTRUCT {
  _:r a :frame-Residence-reside.v 
  _:r :fe-Residence-Resident ?S .  
  _:r :fe-Residence-Location ?O .  
  } WHERE {
  }
  ```

- **Precision:** 79%
Conclusion & Future Work

- We create complex mappings between properties in external KBs and “reified” property-frame-property patterns in FrameBase.

- Future work:
  - Combining with traditional one-to-one mappers (class-class, property-property)
    - This produces transitive complex maps between arbitrary external KBs
  - More very-complex maps
    - (becomes/seems Adj → Noun → Verb)
Conclusion & Future Work

- Web interface for semi-automatic integration (IJCAI 16 demo)
Questions

More information at  http://framebase.org