Expanding the YAGO knowledge base

Thomas Rebele

Télécom ParisTech

2018-07-19
What is a knowledge base?

- Mileva Marić married Albert Einstein
- Albert Einstein has advisor Alfred Kleiner
- Alfred Kleiner won prize Nobel Prize in Physics
- Albert Einstein won prize Nobel Prize in Physics
- Mileva Marić married Alfred Kleiner
What is a knowledge base?

Applications of knowledge bases:
- Question answering
- Semantic search
- Text analysis
- Machine translation
What is YAGO?

- Knowledge base with 10 million entities and >210 million facts
- Automatically extracted from Wikipedia, Wordnet, and Geonames
- Multilingual facts from 10 languages
- Focus on precision
- Developed by Max-Planck Institute for Informatics and Télécom ParisTech
What is YAGO?

Albert Einstein

Albert Einstein was a physicist. His work influenced the philosophy of science. He developed the theory of relativity.

Languages

- English
- German
- French

Categories: Nobel laureates in Physics

- Spouse(s): Mileva Marić
- Doctoral advisor(s): Alfred Kleiner

YAGO

select knowledge

automatic extraction

- Mileva Marić
- married
- Albert Einstein
- has advisor
- Alfred Kleiner
- won prize
- Nobel Prize in Physics

The YAGO knowledge base

What is a knowledge base?

What is YAGO?

Involvement

Outline

Using YAGO for the humanities

Adding Words to Regexes

Answering Queries with Unix Shell

Conclusion
Involvement

- I joined the project in 2014
- Maintenance and development
- Coordinated / contributed to the evaluation
  - 98% facts of the sample were correct

Publication: ISWC 2016 (resource paper)

YAGO is very accurate. But how complete is it?
Contributions:

Extracting more information about residences, gender, birth and death dates

Repairing regular expressions by adding missing words

Preprocessing tabular data by transforming queries to Bash scripts
Contributions:

Extracting more information about residences, gender, birth and death dates

Repairing regular expressions by adding missing words

Preprocessing tabular data by transforming queries to Bash scripts
Every person lives somewhere, but YAGO knows the residence only for 30% of the people.

Every person has a gender, but YAGO knows the gender only for 64% of the people.

How can we make YAGO more complete?
Using YAGO for the humanities: Place of residence

Plato

Plato was a philosopher. He founded the Academy in Athens. He laid the foundation for philosophy.

Categories: 420s BC births | 340s BC deaths | Greek philosopher | Greek male wrestler | Austrian writer

Birthplace: Athens

Previous approach

Plato

Residence: Greece
Using YAGO for the humanities: Place of residence

Plato

Plato was a philosopher. He founded the Academy in Athens. He laid the foundation for philosophy.

Categories: 420s BC births | 340s BC deaths | Greek philosopher | Greek male wrestler | Austrian writer

Birthplace: Athens

previous approach

residence

Greece
Austria
Greece

mapping of 5900 demonyms

Categories: 420s BC births | 340s BC deaths | Greek philosopher | Greek male wrestler | Austrian writer

Greece
Austria
Greece
Using YAGO for the humanities: Place of residence

Plato was a philosopher. He founded the Academy in Athens. He laid the foundation for philosophy.

Categories: 420s BC births | 340s BC deaths | Greek philosopher | Greek male wrestler | Austrian writer

Plato

Birthplace: Athens

Greece: 2
Austria: 1

mapping of 5900 demonyms

previous approach

residence

Greece: 2
Austria: 1

Evaluation

Births per month
Life span over time
Relative population size
Summary

Adding Words to Regexes

Answering Queries with Unix Shell

Conclusion

The YAGO knowledge base

Outline

Using YAGO for the humanities

Place of residence

Gender

Evaluation

Births per month
Life span over time
Relative population size
Summary

8/41
Using YAGO for the humanities: Place of residence

Plato

Plato was a philosopher. He founded the Academy in Athens. He laid the foundation for philosophy.

Categories: 420s BC births | 340s BC deaths | Greek philosopher | Greek male wrestler | Austrian writer

Birthplace: Athens

Previous approach:

Greece | Austria | Greece

Mapping of 5900 demonyms:

Greece: 2
Austria: 1
Using YAGO for the humanities: Gender

Extract gender:

Languages
- English
- German
- French

Albert Einstein

Albert Einstein was a physicist. His work influenced the philosophy of science. He developed the theory of relativity.

Categories: Male scientist | Swiss physicists

From pronoun:
- YAGO’s original algorithm
- Count pronouns (he, him / she, her)
- Assign gender accordingly
Using YAGO for the humanities: Gender

Albert Einstein

Albert Einstein was a physicist. His work influenced the philosophy of science. He developed the theory of relativity.

Categories: Male scientist | Swiss physicists

Languages

English
German
French

From category

From pronoun:

- YAGO’s original algorithm
- Count pronouns (he, him / she, her)
- Assign gender accordingly
Using YAGO for the humanities: Gender

Extract gender:

- **Languages**
  - English
  - German
  - French

Albert Einstein

Albert Einstein was a physicist. His work influenced the philosophy of science. He developed the theory of relativity.

Categories: Male scientist | Swiss physicists

From category

From first name:
- Count males/females for each first name
- Assign names to gender accordingly

From pronoun:
- YAGO’s original algorithm
- Count pronouns (he, him / she, her)
- Assign gender accordingly
Using YAGO for the humanities: Evaluation

- Compare extraction process on Wikipedia dump from 2017-02-20
- Extracted on 11 languages
- Evaluate precision based on a sample of 100 people

<table>
<thead>
<tr>
<th>Extraction</th>
<th>YAGO before</th>
<th>Recall</th>
<th>YAGO now</th>
<th>Recall</th>
<th>Precision</th>
<th>DBpedia (en)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place of residence</td>
<td>0.7m</td>
<td>30%</td>
<td>2.1m</td>
<td>91% (+201%)</td>
<td>97% (*)</td>
<td>0.7m</td>
</tr>
<tr>
<td>Gender</td>
<td>1.5m</td>
<td>64%</td>
<td>2.0m</td>
<td>87% (+35%)</td>
<td>98%</td>
<td>4k</td>
</tr>
<tr>
<td>Birth dates</td>
<td>1.6m</td>
<td>69%</td>
<td>1.7m</td>
<td>74% (+8%)</td>
<td>100%</td>
<td>0.8m</td>
</tr>
<tr>
<td>Death dates</td>
<td>0.7m</td>
<td>33%</td>
<td>0.8m</td>
<td>36% (+10%)</td>
<td>100%</td>
<td>0.3m</td>
</tr>
</tbody>
</table>

Table: Coverage and precision of our methods. Recall relative to total number of people in YAGO (2.2m).

m million       k thousand
(*) 6% of anachronistic residencies (e.g., German Empire instead of Germany)
Using YAGO for the humanities: Births per month

Figure: Births per month in the United States between 2003 and 2015 (with the Student’s t confidence interval at $\alpha = 95\%$).
Using YAGO for the humanities: Births per month

Figure: Births per month in the United States between 2003 and 2015 (with the Student’s t confidence interval at $\alpha = 95\%$).

Relative age effect

The relative age effect describes a bias. People born early in the selection period of sports or academia are more likely to succeed.

Languages

- English
- Euskara

Categories: Ageism | Epidemiology
Figure: Births per month in the United States between 2003 and 2015 (with the Student’s t confidence interval at $\alpha = 95\%$).
Using YAGO for the humanities: Life span over time

Figure: Median age over time, by year of birth

*Figure:* Median age over time, by year of birth.
Using YAGO for the humanities: Relative population size

**Figure:** Relative population size, by century. The y-axis is scaled by a quadratic function.
Using YAGO for the humanities: Summary

- Extension of YAGO:
  - More people with residences (+201%, 97% precision)
  - More people with genders (+35%, 98% precision)
  - More birth and death dates (+8%/10%, 100% precision)

- Case studies:
  - Births per month
  - Life span over time
  - Relative population size over time

- Interdisciplinary project

Publication: ISWC 2017 (workshop paper)

We often had to repair regular expressions (e.g., for matching dates). Can we automate this step?
Using YAGO for the humanities: Summary

Contributions:

- Extracting more information about residences, gender, birth and death dates
- Repairing regular expressions by adding missing words
- Preprocessing tabular data by transforming queries to Bash scripts
Why does YAGO not know the ISBN numbers of my books?

- We want to find ISBN numbers in Wikipedia to include it in YAGO
- We try the regex
  
  $\text{ISBN}(978|979)\ ?\d\{10\}$
Adding Words to Regexes: Introduction

Why does YAGO not know the ISBN numbers of my books?

- We want to find ISBN numbers in Wikipedia to include it in YAGO.
- We try the regex: `ISBN(978|979)?\d{10}`
- Why does the regex not find `I978-2-1234-5680-3`?
- How can we modify the regex automatically to match the word?
Problem statement, first try:

Given
- a regular expression $r$ and
- a set of positive examples $E^+$,

find a regular expression $r'$ such that
- $L(r) \subseteq L(r')$
- $E^+ \subseteq L(r')$
Problem statement, first try:

Given

- a regular expression $r$ and
- a set of positive examples $E^+$,

find a regular expression $r'$ such that

- $L(r) \subseteq L(r')$
- $E^+ \subseteq L(r')$

Solution:

$r' = .*$
Problem statement:

Given

◮ a regular expression $r$,
◮ a set of positive examples $E^{+}$,
◮ a set of negative examples $E^{-}$,

find a regular expression $r'$ such that

◮ $L(r) \subseteq L(r')$
◮ $E^{+} \subseteq L(r')$
◮ $L(r') \cap E^{-}$ is small

ISBN(978|979)？\d{10}

{ I978-2-1234-5680-3 }
{ 0612345678 }
Adding Words to Regexes: Problem statement

Problem statement:

Given

- a regular expression $r$,
- a set of positive examples $E^+$,
- a set of negative examples $E^-$,

find a regular expression $r'$ such that

- $L(r) \subseteq L(r')$
- $E^+ \subseteq L(r')$
- $L(r') \cap E^-$ is small

Evaluation:

- Precision of $r' \geq$ or $\approx$ precision of $r$
- Recall of $r' \geq$ recall of $r$
  (w.r.t. the intended meaning of the regex)
Adding Words to Regexes: What is new in our approach

Previous approaches:

\[ \text{regex} + E^+ + E^- \rightarrow \text{regex} \]

Our approach:

\[ \text{regex} + E^+ + E^- \rightarrow \text{regex} \]

Rationale: creating a large set of positive examples is difficult
Adding Words to Regexes: Approximate regex matching

Step 1: match string and regex approximately [Myers et al. 1989]
Adding Words to Regexes: Finding the gaps

Step 2: find the gaps
- Between regex leaves
Adding Words to Regexes: Finding the gaps

Step 2: find the gaps

- Between regex leaves
- Between characters of the string

```
?...
```

```
I S B N 9 7 8 9 7 9 \d \d ... \d \d
```

```
I 9 7 8 - 2 1 2 3 4 - 5 6 8 0 - 3
```
Step 3 (simple approach): adapt regex, so that it includes the missing parts

```
I 9 7 8 - 2 - 1 2 3 4 - 5 6 8 0 - 3
```

```
I ? | 9 7 8 9 7 9 
S B N 9 7 8 9 7 9
```
Step 3 (simple approach): adapt regex, so that it includes the missing parts.
Adding Words to Regexes: Add missing parts

Step 3 (simple approach): adapt regex, so that it includes the missing parts

---

Outline

Introduction
Problem statement
What is new in our approach
Approximate regex matching
Finding the gaps
Add missing parts
Feedback function
Experiments
Summary
Answering Queries with Unix Shell
Conclusion

---
Step 3 (simple approach): adapt regex, so that it includes the missing parts
Adding Words to Regexes: Add missing parts

Step 3 (adaptive approach): adapt regex, so that it includes the missing parts

Exemplarily for a concatenation:

\[
\{g_1, g_2, g_3\}
\]

\[
\begin{array}{c}
\text{ \quad a \quad b \quad \ldots \quad c \quad d} \\
\quad \uparrow \quad \uparrow \\
\text{\quad \quad g_1 \quad \quad \quad g_2^s} \\
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{ \quad a \quad b \quad \ldots \quad c \quad d} \\
\quad \uparrow \\
? \\
\end{array}
\]

\[
\begin{array}{c}
\{g_1, g_2^s\} \quad \{g_2^e, g_3\}
\end{array}
\]
Step 3 (adaptive approach): adapt regex, so that it includes the missing parts

Exemplarily for a concatenation:

\[
\{g_1, g_2, g_3\}
\]

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\ldots \\
\text{c} \\
\text{d}
\end{array}
\]

\[
\begin{array}{c}
g_1 \\
g_2^s \\
g_2^e \\
g_3
\end{array}
\]

\[
\begin{array}{c}
g_1, g_2^s \\
g_2^e, g_3
\end{array}
\]

\[
\begin{array}{c}
g_2^s \\
\ldots \\
g_2^e \\
g_3
\end{array}
\]

\[
\begin{array}{c}
g_1, g_2^s \\
\text{a} \\
\text{b} \\
\ldots \\
\text{c} \\
\text{d}
\end{array}
\]

\[
\begin{array}{c}
g_2^s, g_3
\end{array}
\]
Adding Words to Regexes: Feedback function

Now we want to find URLs:

► We try regex \( r = \text{http://[a-zA-Z.\-]+} \)
► It does not find \( s = \text{wikipedia.org} \)
► Repaired regex \( r' = (\text{http://})?[a-zA-Z.\-]+ \)

Problem:

► \( r' \) finds all words
► Precision drops
Adding Words to Regexes: Feedback function

Now we want to find URLs:

- We try regex \( r = \text{http://[a-zA-Z.]+} \)
- It does not find \( s = \text{wikipedia.org} \)
- Repaired regex \( r' = (\text{http://})?[a-zA-Z.]+ \)

Problem:

- \( r' \) finds all words
- Precision drops

Solution: use feedback on set of negative examples \( E^- \)

- Determine the parts of the regex that we can make optional
- We use the number of false positives, i.e.,

\[
    f(r') = |E^- \cap L(r')| \leq \alpha |E^- \cap L(r)|
\]

- If \( f(r') = \text{false} \), add the word as disjunction instead: \( \text{http://[a-zA-Z.]+}|\text{wikipedia.org} \)
Input data:

- Datasets:
  - ReLIE [Li et al., 2008],
  - Enron [Babbar et al., 2010], and
  - Wikipedia infobox attributes
- In total 8 tasks (e.g., phone numbers, software names, dates)
- In total 52 regexes
Adding Words to Regexes: Experiments

Input data:
- Datasets: ReLIE [Li et al., 2008], Enron [Babbar et al., 2010], and Wikipedia infobox attributes
- In total 8 tasks (e.g., phone numbers, software names, dates)
- In total 52 regexes

Approaches:
- Dis: $r | s_1 | \cdots | s_n$
- Star: .*
- B&S: [Babbar et al., 2010] (reimplementation)
- Simple
- Adaptive

<table>
<thead>
<tr>
<th>measure</th>
<th>baseline</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>original</td>
<td>dis</td>
<td>star</td>
<td>B&amp;S</td>
<td>simple</td>
<td>$\alpha = 1.0$</td>
<td>$\alpha = 1.1$</td>
</tr>
<tr>
<td>F1</td>
<td>55</td>
<td>55</td>
<td>21</td>
<td>40</td>
<td>56</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>recall</td>
<td>66</td>
<td>67</td>
<td>62</td>
<td>35</td>
<td>69</td>
<td>75</td>
<td>76</td>
</tr>
<tr>
<td>precision</td>
<td>64</td>
<td>64</td>
<td>14</td>
<td>71</td>
<td>64</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>length</td>
<td>56</td>
<td>270</td>
<td>2</td>
<td>3929</td>
<td>250</td>
<td>76</td>
<td>80</td>
</tr>
</tbody>
</table>

Table: Averaged measures for the different systems. Length is # of characters of the regex.
Adding Words to Regexes: Summary

Summary:
- Algorithm for adding missing words to regexes
- Increases recall, while keeping precision stable
- Source code available at https://github.com/thomasrebele/regex-repair

Future work:
- Decrease dependency on $E^-$
- Add a generalization step as postprocessing

Publications: ISWC 2017 (demo), PAKDD 2018 (full paper)

Now that we have all this data, how can we process it efficiently?
Adding Words to Regexes: Summary

Contributions:

Extracting more information about residences, gender, birth and death dates

Repairing regular expressions by adding missing words

Preprocessing tabular data by transforming queries to Bash scripts
Answering Queries with Unix Shell: Motivation

How can I find all my academic ancestors?

Albert Einstein (hasAdvisor) Alfred Kleiner (hasAdvisor) Johann Müller

Relativity (teaches) Statistical physics (teaches) Electromagnetism (teaches)
Answering Queries with Unix Shell: Idea

- SPARQL / OWL
- TSV/n-triples
- database
- result
Answering Queries with Unix Shell: Idea

- SPARQL / OWL
- TSV/n-triples
- database
- result

The YAGO knowledge base
Using YAGO for the humanities
Adding Words to Regexes
Answering Queries with Unix Shell
Motivation
Idea
Approach
Optimization
Experiments
Summary
Conclusion
Answering Queries with Unix Shell: Idea

1. Datalog
2. Algebra
3. Optimize
4. Translate

database

TSV/n-triples

SPARQL / OWL

Bash script

result

©2018 WoTlePRO, Inc.  All rights reserved.
Answering Queries with Unix Shell: Idea

1. Datalog
   - SPARQL / OWL

2. Algebra
   - $\sigma \pi$
   - TSV/n-triples

3. Optimize
   - YAGO

4. Translate
   - Bash script

Database -> Result
Query "Who are Einstein’s academic ancestors?" in SPARQL:

```sparql
SELECT ?Y WHERE {
  <Einstein> <hasAdvisor>+ ?Y
}
```

Translating the query to Datalog (simplified):

```datalog
facts(X, Y, Z) :- read_ntriples input
adv(X, Y) :- facts(X, "hasAdvisor", Y).
result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).
```
Expanding the YAGO knowledge base

Rebele

The YAGO knowledge base
Outline
Using YAGO for the humanities
Adding Words to Regexes
Answering Queries with Unix Shell
Motivation
Idea
Approach
Optimization
Experiments
Summary
Conclusion

Answering Queries with Unix Shell: Approach

\[
\begin{align*}
\text{facts}(X, Y, Z) & :\sim \text{read_ntriples input} \\
\text{adv}(X, Y) & : \text{facts}(X, "hasAdvisor", Y) \\
\text{result}(Y) & : \text{adv("Einstein", Y)} \\
\text{result}(Y) & : \text{result}(X), \text{adv}(X, Y)
\end{align*}
\]

\[
\begin{align*}
\mu_x \\
\bigcup
\begin{align*}
\pi_2 \\
\sigma_{1=Einstein}
\end{align*}
\begin{align*}
\pi_3 \\
\mathcal{A}_{1=1}
\end{align*}
\begin{align*}
\pi_{1,3} \\
\sigma_{2=\text{hasAdvisor}}
\end{align*}
\begin{align*}
x \\
\pi_{1,3} \\
\sigma_{2=\text{hasAdvisor}}
\end{align*}
\begin{align*}
\text{input}
\end{align*}
\text{input}
\end{align*}
\]
Answering Queries with Unix Shell: Approach

1. Datalog
   facts(X, Y, Z) :- read_ntriples input

2. Algebra
   adv(X, Y) :- facts(X, "hasAdvisor", Y).

3. Optimize
   result(Y) :- adv("Einstein", Y).

4. Translate
   result(Y) :- result(X), adv(X, Y).

\[
\mu_X \bigcup_{\pi_2} \pi_3
\]

\[
\sigma_1 = \text{Einstein} \\
\pi_{1,3}
\]

\[
\sigma_2 = \text{hasAdvisor} \\
\pi_{1,3} \leftarrow \text{projection}
\]

\[
\sigma_2 = \text{hasAdvisor} \\
\pi_{1,3} \leftarrow \text{selection}
\]
Expanding the YAGO knowledge base

Rebele

Outline

Using YAGO for the humanities

Adding Words to Regexes

Answering Queries with Unix Shell

Motivation

Idea

Approach

Optimization

Experiments

Summary

Conclusion

Answering Queries with Unix Shell: Approach

1. Datalog
2. Algebra
3. Optimize
4. Translate

Bash script

SPARQL / OWL

facts(X, Y, Z) :- read_ntriples input
adv(X, Y) :- facts(X, "hasAdvisor", Y).
result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).

μ

∪

π₂

σ₁=Einstein

π₁,3

σ₂=hasAdvisor

input

π₃

σ₂=hasAdvisor

input

projection

selection

x

∪

π₁,3

σ₁=1

π₁,3

1

projection

selection
Answering Queries with Unix Shell: Approach

1. Datalog
   \( \sigma \pi \)

2. Algebra
   \( \omega \theta \)

3. Optimize
   \( \mu_x \)

4. Translate
   \( \nu_{1=1} \), join

Using YAGO for the humanities
Adding Words to Regexes
Answering Queries with Unix Shell
Motivation
Idea
Approach
Optimization
Experiments
Summary
Conclusion
Answering Queries with Unix Shell: Approach

facts(X, Y, Z) :- read_ntriples input

adv(X, Y) :- facts(X, "hasAdvisor", Y).

result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).

μₓ ← ⋓ least fixed point

recursive call

σ₁=Einstein

π₁,3

σ₂=hasAdvisor

input

σ₂=hasAdvisor

input
Answering Queries with Unix Shell: Approach

```
facts(X, Y, Z) :- read_ntriples input

adv(X, Y) :- facts(X, "hasAdvisor", Y).

result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).
```

\[
\begin{align*}
\mu & \quad \bigcup \\
\pi_2 & \quad \sigma_1 = \text{Einstein} \\
\pi_1,3 & \quad \pi_1,3 \\
\sigma_2 = \text{hasAdvisor} & \quad \sigma_2 = \text{hasAdvisor} \\
(\text{input}, \text{s.}, \text{Einstein}) & \quad (\text{input}, \text{s.}, \text{Einstein}) \\
(\text{Einstein}, \text{adv}) & \quad (\text{Einstein}, \text{adv})
\end{align*}
\]
Answering Queries with Unix Shell: Approach

\[
\begin{align*}
facts(X, Y, Z) & :\sim \text{read_ntriples input} \\
adv(X, Y) & :\sim \text{facts}(X, "hasAdvisor", Y). \\
result(Y) & :\sim \text{adv}("Einstein", Y). \\
result(Y) & :\sim \text{result}(X), \text{adv}(X, Y). 
\end{align*}
\]

\[
\begin{align*}
\mu_x \\
\cup \\
\pi_2 \\
\pi_3 \\
\sigma_{1=Einstein} \\
\pi_{1,3} \\
\sigma_{2=\text{hasAdvisor}} \\
\text{input} \\
\sigma_{2=\text{hasAdvisor}} \\
\text{input}
\end{align*}
\]
Answering Queries with Unix Shell: Approach

```
facts(X, Y, Z) :- read_ntriples input
adv(X, Y) :- facts(X, "hasAdvisor", Y).
result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).
```

**Diagram:**
- **Datalog:** \( \sigma \pi \)
- **Algebra:** \( \mu_x \)
- **Optimize:** \( \pi_2 \cup \pi_3 \)
- **Translate:** \( \pi_{1,3} \)
- **Input:** \( \sigma_2 = \text{hasAdvisor} \)
- **Result:** \( \sigma_2 = \text{hasAdvisor} \)

---

**Outline:**
- Using YAGO for the humanities
- Adding Words to Regexes
- Answering Queries with Unix Shell

**Motivation**

**Idea**

**Approach**

**Optimization**

**Experiments**

**Summary**

**Conclusion**
Answering Queries with Unix Shell: Approach

<table>
<thead>
<tr>
<th>SPARQL / OWL</th>
<th>( \sigma \pi )</th>
<th>Bash script</th>
</tr>
</thead>
<tbody>
<tr>
<td>facts(X, Y, Z) :- read_ntriples input</td>
<td>( \mu_x )</td>
<td></td>
</tr>
<tr>
<td>adv(X, Y) :- facts(X, &quot;hasAdvisor&quot;, Y).</td>
<td>( \bigcup )</td>
<td></td>
</tr>
<tr>
<td>result(Y) :- adv(&quot;Einstein&quot;, Y).</td>
<td>( \pi_2 )</td>
<td></td>
</tr>
<tr>
<td>result(Y) :- result(X), adv(X, Y).</td>
<td>( \pi_3 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \sigma_1\text{=}\text{Einstein} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \pi_{1,3} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \sigma_2\text{=}\text{hasAdvisor} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>input</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Expanding the YAGO knowledge base

The YAGO knowledge base

Outline

Using YAGO for the humanities

Adding Words to Regexes

Answering Queries with Unix Shell

Motivation

Idea

Approach

Optimization

Experiments

Summary

Conclusion

Answering Queries with Unix Shell: Approach

1. Datalog

2. Algebra

3. Optimize

4. Translate

Bash script

\[ \text{facts}(X, Y, Z) \triangleright \text{read_ntriples input} \]

\[ \text{adv}(X, Y) \triangleright \text{facts}(X, "hasAdvisor", Y). \]

\[ \text{result}(Y) \triangleright \text{adv}("Einstein", Y). \]

\[ \text{result}(Y) \triangleright \text{result}(X), \text{adv}(X, Y). \]

\[ \mu \]

\[ \pi \]

\[ \sigma \]

\[ \sigma_1 = \text{Einstein} \]

\[ \sigma_2 = \text{hasAdvisor} \]

\[ \pi \]

\[ \pi_1, 3 \]

\[ \pi_2 \]

\[ \pi_3 \]

\[ x \]

\[ \sigma_2 = \text{hasAdvisor} \]

\[ \text{input} \]
Answering Queries with Unix Shell: Approach

```
facts(X, Y, Z) :- read_ntriples input

adv(X, Y) :- facts(X, "hasAdvisor", Y).

result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).
```
Answering Queries with Unix Shell: Approach

facts(X, Y, Z) :- read_ntriples input

adv(X, Y) :- facts(X, "hasAdvisor", Y).

result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).
Answering Queries with Unix Shell: Approach

facts(X, Y, Z) :- read_ntriples input

adv(X, Y) :- facts(X, "hasAdvisor", Y).

result(Y) :- adv("Einstein", Y).
result(Y) :- result(X), adv(X, Y).
Answering Queries with Unix Shell: Approach

Optimizations:

\[
\begin{align*}
\mu_x & \cup \\
\pi_2 & | \quad \pi_3 \\
\sigma_{1=Einstein} & | \quad \neq_{1=1} \\
\pi_{1,3} & | \quad \pi_{1,3} \\
\sigma_{2=\text{hasAdvisor}} & | \quad \sigma_{2=\text{hasAdvisor}} \\
\text{input} & | \quad \text{input} \quad \text{input} \\
\end{align*}
\]
Optimizations:

\[
\mu_x \
\cup \:\pi_2 \
\sigma_1=Einstein \
\pi_{1,3} \sigma_2=\text{hasAdvisor} \input
\]

\[
\mu_x \
\cup \:\pi_3 \
\sigma_1=Einstein \
\pi_{1,3} \sigma_2=\text{hasAdvisor} \delta_x \input
\]
Answering Queries with Unix Shell: Approach

Optimizations:

\[
\begin{align*}
\mu_x & \\
\cup & \\
\pi_2 & \pi_3 \\
\sigma_1 & = \text{Einstein} \\
\pi_{1,3} & x \pi_{1,3} \\
\sigma_2 & = \text{hasAdvisor} \\
\text{input} & \text{input}
\end{align*}
\]

\[
\begin{align*}
\mu_x & \\
\text{sort} & \\
\pi_3 & \\
\sigma_1 & = \text{Einstein} \\
\pi_{1,3} & \\
\sigma_2 & = \text{hasAdvisor} \\
\text{input} & \\
\delta_x & \pi_{1,3} \\
\sigma_2 & = \text{hasAdvisor} \\
\text{input}
\end{align*}
\]
Answering Queries with Unix Shell: Approach

Optimizations:

```
\begin{align*}
\mu_x & \cup \\
\pi_2 & \pi_3 \\
\sigma_1 &= Einstein & \varnothing_1 &= 1 \\
\pi_{1,3} & x & \pi_{1,3} \\
\sigma_2 &= hasAdvisor & \sigma_2 &= hasAdvisor \\
\text{input} & \text{input} \\
\end{align*}
```

```
\begin{align*}
\mu_x & \cup \\
\pi_3 & \text{sort} \\
\sigma_1 &= Einstein & \pi_3 & \text{sort} \\
\sigma_2 &= hasAdvisor & \varnothing_1 &= 1 \\
\text{input} & \text{sort}_1 & \text{sort}_1 \\
\delta_x & \pi_{1,3} & \sigma_2 &= hasAdvisor \\
& \text{input} \\
\end{align*}
```
Answering Queries with Unix Shell: Approach

Optimizations:

\[
\mu_x \\
\bigcup \\
\pi_2 \\
\sigma_1 = \text{Einstein} \\
\pi_1,3 \\
\sigma_2 = \text{hasAdvisor} \\
\text{input}
\]

\[
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\text{input}
\]

\[
\sigma_1 = \text{Einstein} \\
\pi_1,3 \\
\sigma_2 = \text{hasAdvisor} \\
\text{input}
\]

\[
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\text{input}
\]

\[
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\sigma_1 = \text{Einstein} \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]

\[
\mu_x \\
\bigcup \\
\pi_2 \\
\sigma_1 = \text{Einstein} \\
\pi_1,3 \\
\sigma_2 = \text{hasAdvisor} \\
\text{input}
\]

\[
\sigma_1 = \text{Einstein} \\
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]

\[
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\sigma_1 = \text{Einstein} \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]

\[
\mu_x \\
\bigcup \\
\pi_2 \\
\sigma_1 = \text{Einstein} \\
\pi_1,3 \\
\sigma_2 = \text{hasAdvisor} \\
\text{input}
\]

\[
\sigma_1 = \text{Einstein} \\
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]

\[
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\sigma_1 = \text{Einstein} \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]

\[
\mu_x \\
\bigcup \\
\pi_2 \\
\sigma_1 = \text{Einstein} \\
\pi_1,3 \\
\sigma_2 = \text{hasAdvisor} \\
\text{input}
\]

\[
\sigma_1 = \text{Einstein} \\
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]

\[
\pi_3 \\
\sigma_2 = \text{hasAdvisor} \\
\pi_1,3 \\
\sigma_1 = \text{Einstein} \\
\delta_x \\
\text{sort_1} \\
\text{sort}
\]
Answering Queries with Unix Shell: Approach

awk '(
    \$1 == "Einstein"
    \&\& \$2 == "hasAdvisor"
) {
    print \$3 \n    }' <(read_ntriples input)

# lock a
(
sort -k 1 pre_a > a
    # unlock a
) &
Answering Queries with Unix Shell: Approach

1. Datalog
2. Algebra
3. Optimize
4. Translate

Bash script

awk '  
(   $1 == "Einstein" 
    && $2 == "hasAdvisor")  
    { print $3 >> "b" }  
($2 == "hasAdvisor")  
    { print $1 FS $3 >> "pre_a" }  
, <(read_ntriples input)

# lock a
(  
    sort -k 1 pre_a > a  
    # unlock a
) &
Answering Queries with Unix Shell: Approach

awk '    
    ($1 == "Einstein" && $2 == "hasAdvisor")
    { print $3 >> "b" }    
    ($2 == "hasAdvisor")
    { print $1 FS $3 >> "pre_a" }    
    ', <(read_ntriples input)

# lock a

( sort -k 1 pre_a > a
  # unlock a
) &
Answering Queries with Unix Shell: Approach

awk '  
    (  
        $1 == "Einstein"  
        && $2 == "hasAdvisor"  
    )  
    { print $3 >> "b" }  
    ($2 == "hasAdvisor")  
    { print $1 FS $3 >> "pre_a" }  
    ) <(read_ntriples input)

# lock a
(
    sort -k 1 pre_a > a  
    # unlock a
) &
Answering Queries with Unix Shell: Approach

```
awk ' (
  $1 == "Einstein"
  && $2 == "hasAdvisor")
  { print $3 >> "b" }
($2 == "hasAdvisor")
  { print $1 FS $3 >> "pre_a" }
  <(read_ntriples input)

# lock a
( (sort -k 1 pre_a > a
  # unlock a
) &
```

1. Datalog
2. Algebra
3. Optimize
4. Translate

SPARQL / OWL → Bash script

Outline
- Using YAGO for the humanities
- Adding Words to Regexes
- Answering Queries with Unix Shell

Motivation
- Idea
- Approach
- Optimization
- Experiments

Summary

Conclusion
Answering Queries with Unix Shell: Approach

while
  # ...
  sort -k 1 -u
  # (wait for a
  join -1 1 -2 1 -o 2.2
  # (sort -k 1 -u delta)
  a
  )
  # ...
  [ -s delta ];
  do continue; done
Answering Queries with Unix Shell: Approach

```
while
  # ...
  sort -k 1 -u
      <( # wait for a
          join -1 1 -2 1 -o 2.2
          <(sort -k 1 -u delta)
          a
      )
  # ...
  [ -s delta ];
  do continue; done
```
Answering Queries with Unix Shell: Optimization

How can I find all professors?

Professor(X) :- Person(X),
              teachesCourse(X,Y).
Professor(X) :- advisorOf(X,Y),
              Professor(Y).
Person(X) :- Employee(X).
Person(X) :- Professor(X).
How can I find all professors?

Professor(X) :- Person(X), teachesCourse(X,Y).
Professor(X) :- advisorOf(X,Y), Professor(Y).

Person(X) :- Employee(X).
Person(X) :- Professor(X).
How can I find all professors?

Professor(X) :- Person(X),
              teachesCourse(X,Y).
Professor(X) :- advisorOf(X,Y),
              Professor(Y).
Person(X) :- Employee(X).
Person(X) :- Professor(X).

Combining the first and the last rule leads to

Professor(X) :- Professor(X),
              teachesCourse(X,Y).
How can I find all professors?

Professor(X) :- Person(X),
              teachesCourse(X,Y).
Professor(X) :- advisorOf(X,Y),
              Professor(Y).
Person(X) :- Employee(X).
Person(X) :- Professor(X).

Combining the first and the last rule leads to

Professor(X) :- Professor(X),
              teachesCourse(X,Y).
Answering Queries with Unix Shell: Optimization

\[
\mu_x \bigcup \pi_1 \ni_{1=1} \bigcup \text{teachesCourse} \quad \bigcup \ni_{2=1} \text{advisorOf} x
\]

employee \( x \)
Answering Queries with Unix Shell: Optimization

\[ \mu_x \]

\[ \bigcup \]

\[ \pi_1 \]

\[ \bigotimes_{1=1} \]

\[ \bigcup \text{teachesCourse} \]

\[ \bigotimes_{2=1} \]

\[ \text{employee} \]

\[ \mu_x \]

\[ \bigcup \]

\[ \pi_1 \]

\[ \bigotimes_{1=1} \]

\[ \bigcup \text{teachesCourse} \]

\[ \bigotimes_{2=1} \]

\[ \text{advisorOf} \]

\[ x \]
Answering Queries with Unix Shell: Optimization
Answering Queries with Unix Shell: Optimization

Expanding the YAGO knowledge base

Outline

Using YAGO for the humanities

Adding Words to Regexes

Answering Queries with Unix Shell

Motivation

Idea

Approach

Optimization

Experiments

Summary

Conclusion
Answering Queries with Unix Shell: Optimization

\[
\mu_x (\pi_1 (\mu_{x} (\pi_1 (\pi_1 (\pi_2 (x))))) \cup (\pi_1 (\pi_2 (x)))) \Rightarrow \text{necessary}
\]

\[
\Rightarrow \text{superfluous}
\]

Outline
- Using YAGO for the humanities
- Adding Words to Regexes
- Answering Queries with Unix Shell
  - Motivation
  - Idea
  - Approach
  - Optimization
  - Experiments
  - Summary
- Conclusion
Answering Queries with Unix Shell: Optimization

\[
\begin{align*}
\forall_{x=1}^{c_1} & \mu_x (\mathcal{Q}) \\
\bigcup (c_1) & \pi_1 (\mathcal{Q}) \\
\bigcup (c_1) & \text{teachesCourse} \\
\bigcup (c_1) & \text{advisorOf} \\
\bigcup (c_1) & \text{employee} \\
\Rightarrow & \text{necessary} \\
\Rightarrow & \text{superfluous}
\end{align*}
\]
Answering Queries with Unix Shell: Experiments

- Dataset: LUBM university benchmark
- 14 different queries
- Competitors: Datalog-based systems (DLV, Souffle, RDFox),
  Triple stores (Jena, Stardog, Virtuoso),
  Database management systems (MonetDB, Postgres)

Number of finished queries within time limit

* = we folded the TBox into the query
Answering Queries with Unix Shell: Experiments

- Bash
- DLV
- RDFox
- Stardog
- Souffle
- Jena
- Virtuoso
- MonetDB
- Postgres

![Graph showing runtime vs. number of universities for various systems.](image-url)
### Answering Queries with Unix Shell: Experiments

<table>
<thead>
<tr>
<th>dataset</th>
<th>Bash</th>
<th>RDFox</th>
<th>BigDatalog</th>
<th>Stardog</th>
<th>Virtuoso</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiveJournal</td>
<td>117</td>
<td>70</td>
<td>532</td>
<td>941</td>
<td>-</td>
</tr>
<tr>
<td>orkut</td>
<td>225</td>
<td>121</td>
<td>1838</td>
<td>1123</td>
<td>-</td>
</tr>
<tr>
<td>friendster</td>
<td>16306</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table:** Runtime for the reachability query, in seconds.
Answering Queries with Unix Shell: Summary

Summary:

- Preprocess large datasets without installing software
- Supports subset of SPARQL / OWL and Datalog as query language
- Try it online at https://www.thomasrebele.org/projects/bashlog
- Source code available at https://github.com/thomasrebele/bashlog

Future work:

- Numerical comparisons
- Aggregations (e.g., max, count)

Publication: ISWC 2018 (full paper)
Conclusion

This thesis showed how to extend YAGO along several axes:

- Improve completeness w.r.t. people
- Automatically repairing of its regular expressions
- Preprocessing queries using only a Bash shell

Other accomplishments:

- Source code of all contributions is available online
- Publications at ISWC 2016 (resource paper), ISWC 2017 (demo, workshop), PAKDD 2018 (full paper), ISWC 2018 (full paper)

Future work:

- More studies on human society using facts from YAGO (ongoing)
- Combine YAGO and Wikidata
- Queries with numerical comparisons and aggregations