CANONICALISATION OF MONOTONE SPARQL QUERIES

JAIME SALAS & AIDAN HOGAN
MOTIVATION
MOTIVATION

- SPARQL is a highly expressive querying language.
- As a result, we can express the same query in different ways.

```sql
SELECT *
WHERE{
    ?actor :acts_in ?film;
    :salary ?wage .
}
```

```sql
SELECT *
WHERE{
    ?actor :acts_in ?movie;
    :salary ?salary .
}
```
MOTIVATION

- SPARQL is a highly expressive querying language.
- As a result, we can express the same query in different ways.
MOTIVATION

```
SELECT DISTINCT ?actor ?actor2
WHERE{
}
```

```
SELECT DISTINCT ?actor ?actor2
WHERE{
}
```
SELECT DISTINCT ?v ?w
WHERE{
    UNION { :a :b :c } UNION { ?x6 ?y6 ?z6 . }
}

SELECT DISTINCT ?n
WHERE{
    { ?n :cousin ?m2 . } UNION { ?x4 ?y4 ?n . }
}

SELECT DISTINCT ?v ?w
WHERE{
}
MOTIVATION

```
SELECT DISTINCT ?v ?w
WHERE{
    UNION { :a :b :c } UNION { ?x6 ?y6 ?z6 .}
}
```

```
SELECT DISTINCT ?n
WHERE{
    { ?n :cousin ?m2 .} UNION { ?x4 ?y4 ?n .}
}
```

```
SELECT DISTINCT ?v ?w
WHERE{
    UNION { ?v :cousin ?x3 .} UNION { ?x6 ?y6 ?z6 .}
}
```
SELECT DISTINCT ?v ?w
WHERE{
    { ?v :cousin ?w . } UNION { ?w :cousin ?v . }
    UNION { a :b :c } UNION { ?x6 ?y6 ?z6 . }
}

SELECT DISTINCT ?v ?w
WHERE{
    { ?v :cousin ?w . } UNION { ?w :cousin ?v . }
}
MOTIVATION

- Query equivalence is a hard problem.
- This produces an overload on SPARQL endpoints.
- We could improve caching systems.
Given a query $Q$:

```sql
SELECT ?game ?release
WHERE{
}
```

Given an RDF dataset $G$:

[Diagram showing relationships between objects: releaseYear, developer, instanceof, videoGame, PokemonYellow, PokemonCrystal, GameFreak, Nintendo, 1996, 1998, 2001.]
The evaluation of $Q$ over $G$, $Q(G)$ would give the following results:

<table>
<thead>
<tr>
<th>Game</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pokemon Yellow</td>
<td>1998</td>
</tr>
<tr>
<td>Pokemon Crystal</td>
<td>2001</td>
</tr>
<tr>
<td>SuperMario64</td>
<td>1996</td>
</tr>
</tbody>
</table>
SELECT DISTINCT ?game ?release
WHERE{
  ?game :developer :GameFreak .
}

SELECT DISTINCT ?game ?release
WHERE{
}
We say that $Q'$ contains $Q$ ($Q \subseteq Q'$) if the evaluation of $Q'$ contains all of the results of the evaluation of $Q$.

<table>
<thead>
<tr>
<th>game</th>
<th>release</th>
</tr>
</thead>
<tbody>
<tr>
<td>:PokemonYellow</td>
<td>1998</td>
</tr>
<tr>
<td>:PokemonCrystal</td>
<td>2001</td>
</tr>
</tbody>
</table>
Given a query Q

```
SELECT DISTINCT ?game ?release
WHERE{
}
```

Given another query Q’

```
SELECT DISTINCT ?game ?release
WHERE{
}
```
We say that $Q'$ is equivalent to $Q$ ($Q \equiv Q'$) if $Q'$ contains $Q$ and $Q$ contains $Q'$.

```sql
SELECT DISTINCT ?game ?release
WHERE{
}
```

```sql
SELECT DISTINCT ?game ?release
WHERE{
}
```
CONGRUENCE

Given a query $Q$

```
SELECT ?game ?release
WHERE{
}
```

Given another query $Q'$

```
SELECT ?title ?year
WHERE{
}
```
Q’ is congruent to Q (Q ≅ Q) iff there exists a bijective variable mapping θ such that Q is equivalent to θ(Q’)

```sparql
SELECT ?game ?release
WHERE{
}
```

```sparql
SELECT ?title ?year
WHERE{
}
```

Where θ is a function that maps ?game to ?title and ?release to ?year.
CONJUNCTIVE QUERIES (CQ)

- These queries contain only conjunctions of triples.

```sparql
SELECT ?game ?release
WHERE{
  ?game :developer :GameFreak .
}
```
UNIONS OF CONJUNCTIVE QUERIES (UCQ)

- These queries contain unions of conjunctive queries.

```sparql
SELECT DISTINCT ?m
WHERE{
}
```
These queries contain only conjunctions and unions.

```
SELECT DISTINCT ?m
WHERE{
    UNION
    UNION
    { ?a :genre :comedy . }
    UNION
    { ?a :genre :horror . }
}
```
## Complexity (Set Semantics)

<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>SPARQL 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>NP-Complete</td>
<td>NP-Complete</td>
<td>PSPACE-Complete</td>
</tr>
<tr>
<td>Containment</td>
<td>NP-Complete</td>
<td>NP-Complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Equivalence</td>
<td>NP-Complete</td>
<td>NP-Complete</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>
## Complexity (Bag Semantics)

<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>SPARQL 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>NP-Complete</td>
<td>NP-Complete</td>
<td>PSPACE-Complete</td>
</tr>
<tr>
<td>Containment</td>
<td>Decidability open</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Equivalence</td>
<td>GI-Complete</td>
<td>GI-Complete</td>
<td>?</td>
</tr>
</tbody>
</table>
OBJECTIVE
- Design a canonicalisation algorithm $CAN(Q)$ such that $CAN(Q) = CAN(Q')$ iff $Q \equiv Q'$. 

```
SELECT DISTINCT ?title ?year
WHERE{
}
```

```
SELECT DISTINCT ?game ?release
WHERE{
  ?game :instanceOf :videoGame;
  :releaseYear ?release .
}
```

```
SELECT DISTINCT ?v0 ?v1
WHERE{
}
```
This algorithm should cover the majority of real-world queries.

It should be efficient for most real-world queries.
PROPOSED SOLUTION
Our algorithm can be divided in the following steps:

- UCQ transformation (Union of Conjunctive Queries)
- Represent the query as an RDF graph.
- Removal of redundancy (minimisation).
- Canonical labelling
- Retrieval of canonical query
\[ A \bowtie B \bowtie C \cup D \]

\[ (A \bowtie B \bowtie C) \cup (A \bowtie B \bowtie D) \]

```
SELECT DISTINCT ?m
WHERE{
    ?a :title ?m .
    { ?a :actor ?p }
UNION
    { ?a :actress ?p }
}
```

```
SELECT DISTINCT ?m
WHERE{
    { ?a :title ?m .
    ?d :directed ?a . }
UNION
    { ?a :title ?m .
    ?d :directed ?a . }
```
- Represent SPARQL queries as RDF graphs.
- Denoted as \textit{r-graphs}.

\begin{verbatim}
SELECT ?game ?release
WHERE{
  ?game :instanceOf :videoGame;
  :releaseYear ?release .
}
\end{verbatim}
- Represent SPARQL queries as RDF graphs.
- Denoted as *r-graphs*.

```
SELECT ?game ?release
WHERE{
  ?game :instanceOf :videoGame;
  :releaseYear ?release .
}
```
This process consists in reducing the query to its minimal form without altering its semantic value.

Only applicable to set semantics. In bag semantics, we want to preserve the multiplicity of results.
Minimisation is crucial for defining a canonical form.
To illustrate, this is the case with numbers:

+02  +02.0  2.000
  2
One of the redundancies we need to remove are redundant triple patterns.

```
SELECT DISTINCT ?game ?release
WHERE{
}
```

```
SELECT DISTINCT ?game ?release
WHERE{
}
```
REDUNDANT TRIPLES
REDUNDANT CONJUNCTIVE QUERIES

```
SELECT DISTINCT ?m
WHERE{
    ?a :title ?m .
}
```

```
SELECT DISTINCT ?m
WHERE{
    ?a :title ?m .
}
```

```
SELECT DISTINCT ?m
WHERE{
    { ?a :title ?m . }
    UNION
    { ?a ?r ?p . }
}
```

```
SELECT DISTINCT ?m
WHERE{
}
```
- We verify if there are conjunctive queries that contain other conjunctive queries.
- If a conjunctive query is contained by another, we keep the one that’s more *general*. 
REdundant Conjunctive Queries
REDUNDANT CONJUNCTIVE QUERIES

?a :title ?m .

?b :title ?m .
REDUNDANT CONJUNCTIVE QUERIES

?a :title ?m .  

?b :title ?m .  
We use an efficient canonical labelling algorithm to determine canonical variable names.
CANONICAL LABELLING
SELECT DISTINCT ?μ
WHERE{
}
SPARQL 1.0 contains more features other than JOIN and UNION:

- OPTIONAL and FILTER.
- Solution modifiers (LIMIT, OFFSET, ORDER BY, etc.)
- Named graphs

```sparql
SELECT ?game ?developer
WHERE{
  OPTIONAL{ ?game :developer ?developer }
  FILTER( ?release > 2000)
}
```

```sparql
SELECT ?game ?release
WHERE{
}
ORDER BY DESC(?release)
LIMIT 10
```
- We cannot guarantee a complete algorithm for queries with these features.
- We *can* guarantee that our solution won’t change the semantic value of the query.
EXPERIMENTS AND RESULTS
EXPERIMENTS AND RESULTS

- Research questions:
  - How is the performance of the canonicalisation algorithm?
  - How many more duplicate queries does it find versus baseline algorithms?
  - Since the algorithm is doubly-exponential in the worst-case scenario, at what point does it begin to fail?
EXPERIMENTS AND RESULTS

- To answer the first question, we use queries from the LSQ dataset.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTINCT</td>
<td>143,522</td>
</tr>
<tr>
<td>JOIN</td>
<td>309,087</td>
</tr>
<tr>
<td>UNION</td>
<td>34,282</td>
</tr>
<tr>
<td>Projection</td>
<td>665,956</td>
</tr>
<tr>
<td>FILTER</td>
<td>181,606</td>
</tr>
<tr>
<td>OPTIONAL</td>
<td>282,700</td>
</tr>
<tr>
<td>Named graph features</td>
<td>234,860</td>
</tr>
<tr>
<td>Solution Modifiers</td>
<td>5,810</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Combination</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>1,480</td>
</tr>
<tr>
<td>UD</td>
<td>1,462</td>
</tr>
<tr>
<td>UJ</td>
<td>1,902</td>
</tr>
<tr>
<td>UDJ</td>
<td>372</td>
</tr>
<tr>
<td>U*</td>
<td>6,127</td>
</tr>
<tr>
<td>UJ*</td>
<td>199</td>
</tr>
<tr>
<td>UD*</td>
<td>168</td>
</tr>
<tr>
<td>UDJ*</td>
<td>22,572</td>
</tr>
</tbody>
</table>
EXPERIMENTS AND RESULTS

- We tested three algorithms:
  - Baseline
  - Labelling
  - Full
## EXPERIMENTS AND RESULTS

- How is the performance of the canonicalisation algorithm?

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (s)</th>
<th>Duplicates</th>
<th>Max duplicates</th>
<th>Total queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>211</td>
<td>3960</td>
<td>12</td>
<td>768618</td>
</tr>
<tr>
<td>Labelling</td>
<td>28066</td>
<td>10722</td>
<td>307</td>
<td>768618</td>
</tr>
<tr>
<td>Full</td>
<td>77023</td>
<td>10722</td>
<td>307</td>
<td>768618</td>
</tr>
</tbody>
</table>
EXPERIMENTS AND RESULTS

- How many more duplicate queries does it find versus baseline algorithms?

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (s)</th>
<th>Duplicates</th>
<th>Max duplicates</th>
<th>Total queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>211</td>
<td>3960</td>
<td>12</td>
<td>768618</td>
</tr>
<tr>
<td>Labelling</td>
<td>28066</td>
<td>10722</td>
<td>307</td>
<td>768618</td>
</tr>
<tr>
<td>Full</td>
<td>77023</td>
<td>10722</td>
<td>307</td>
<td>768618</td>
</tr>
</tbody>
</table>
EXPERIMENTS AND RESULTS
EXPERIMENTS AND RESULTS

- In order to stress-test the algorithm, we designed an experiment where we canonicalize r-graphs of queries based on well-known graphs.
These results show that the algorithm stops working for queries that contain over 1700 triple patterns.
CONCLUSIONS

- We have designed a canonicalisation algorithm for monotone SPARQL queries.
- Results show that the algorithm is efficient for most real-world queries.
- We find more duplicated queries than our baseline method.
- On the other hand, there are presumably no queries with redundancies.
- The results show that the algorithm stops being efficient for queries that contain over 1700 triple patterns.
FUTURE WORK

- Extend the scope of the algorithm to include SPARQL 1.1
- Implement similar techniques for other querying languages.
DEMO (D24)