dRDF: Entailment for Domain-Restricted RDF

Reinhard Pichler ¹  Axel Polleres ²  Fang Wei ¹  Stefan Woltran ¹
¹ Institute for Information Systems, TU Vienna, Austria
² DERI, National University of Ireland, Galway

Alternative subtitles:
Blank nodes are fun (at least for theoreticians)
or
Blank nodes ain’t THAT evil! 😊
RDF Entailment: $G_1 \models G_2$

- Does graph $G_1$ entail $G_2$?

- Boils down to:
  "Is there a blank node renaming $\mu$ for blank nodes in $G_2$ such that $\mu(G_2) \subseteq G_1$"?

- "Folklore": Well-known to be NP-complete (cf. RDF Semantics [Hayes, 2004])

- Observation: Blank nodes are causing the "trouble" of making the problem intractable… ground entailment well known to be in P.

**Starting point for our work:**

Besides completely forbidding blank nodes…

… What else can we do to make this problem tractable?
Restrictions on RDF graphs considered in this paper:

1. **Domain-Restricted Graphs**: Restrict the domain blank nodes can range over to a finite set of objects.

2. **Graphs with Bounded Treewidth**: Restrict the graph structure of RDF graphs: bounded-treewidth (a generalization of acyclicity)

Effects:

1. …OOPS! With *finite domains*, complexity actually jumps from NP to \( \text{coNP}^{\text{NP}} = \Pi_2^p \) 😞
2. Not all is lost: *bounded treewidth* guarantees tractability for general entailment and \( \text{coNP} \) bound for domain-restricted graphs.

Summary:

<table>
<thead>
<tr>
<th></th>
<th>domain-restricted graphs</th>
<th>Unrestricted graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounded treewidth</td>
<td>coNP-complete</td>
<td>in P 😊</td>
</tr>
<tr>
<td>unbounded treewidth</td>
<td>( \Pi_2^p )-complete</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>
# Domain-Restricted Graphs: Example

<table>
<thead>
<tr>
<th>$G_1$</th>
<th>$G_2$</th>
<th>$G_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(__b_1, \text{foaf:name, &quot;Fang&quot;})$, $(__b_2, \text{foaf:name, &quot;Stefan&quot;})$, $(__b_3, \text{foaf:name, &quot;Reini&quot;})$, $(__b_1, \text{worksWith, } __b_2)$, $(__b_2, \text{worksWith, } __b_3)$</td>
<td>$(__b_1, \text{foaf:name, &quot;Stefan&quot;})$, $(__b_2, \text{foaf:name, &quot;Reini&quot;})$, $(__b_3, \text{foaf:name, &quot;Fang&quot;})$, $(__b_1, \text{worksWith, } __b_2)$, $(__b_3, \text{worksWith, } __b_1)$, $(__b_1, \text{worksWith, } __b_3)$, $(__b_4, \text{foaf:name, &quot;Axel&quot;})$, $(__b_1, \text{worksWith, } __b_4)$</td>
<td>$(__b_2, \text{foaf:name, &quot;Stefan&quot;})$, $(__b_1, \text{foaf:name, &quot;Axel&quot;})$, $(__b_2, \text{worksWith, } __b_1)$</td>
</tr>
</tbody>
</table>
## Domain-Restricted Graphs: Example

<table>
<thead>
<tr>
<th>$G_1$</th>
<th>$G_2$</th>
<th>$G_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(:b₁, foaf:name, &quot;Fang&quot;),</td>
<td>(:b₁, foaf:name, &quot;Stefan&quot;),</td>
<td>(:b₂, foaf:name, &quot;Stefan&quot;),</td>
</tr>
<tr>
<td>(:b₂, foaf:name, &quot;Stefan&quot;),</td>
<td>(:b₂, foaf:name, &quot;Reini&quot;),</td>
<td>(:b₁, foaf:name, &quot;Axel&quot;),</td>
</tr>
<tr>
<td>(:b₃, foaf:name, &quot;Reini&quot;),</td>
<td>(:b₃, foaf:name, &quot;Fang&quot;),</td>
<td>(:b₁, foaf:name, &quot;Axel&quot;),</td>
</tr>
<tr>
<td>(:b₁, worksWith, :b₂),</td>
<td>(:b₁, worksWith, :b₂),</td>
<td>(:b₃, worksWith, :b₁),</td>
</tr>
<tr>
<td>(:b₂, worksWith, :b₃)</td>
<td></td>
<td>(:b₂, worksWith, :b₁)</td>
</tr>
</tbody>
</table>

![Diagram](image)
## Domain-Restricted Graphs: Example

<table>
<thead>
<tr>
<th></th>
<th>G₁</th>
<th>G₂</th>
<th>G₃</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(_:b₁, foaf:name, &quot;Fang&quot;),</td>
<td>(_:b₁, foaf:name, &quot;Stefan&quot;),</td>
<td>(_:b₂, foaf:name, &quot;Stefan&quot;),</td>
</tr>
<tr>
<td></td>
<td>(_:b₂, foaf:name, &quot;Stefan&quot;),</td>
<td>(_:b₂, foaf:name, &quot;Reini&quot;),</td>
<td>(_:b₁, foaf:name, &quot;Axel&quot;),</td>
</tr>
<tr>
<td></td>
<td>(_:b₃, foaf:name, &quot;Reini&quot;),</td>
<td>(_:b₃, foaf:name, &quot;Fang&quot;),</td>
<td>(_:b₂, :worksWith, _:b₁)</td>
</tr>
<tr>
<td></td>
<td>(_:b₁, :worksWith, _:b₂),</td>
<td>(_:b₁, :worksWith, _:b₃),</td>
<td>(_:b₁, :worksWith, _:b₄)</td>
</tr>
<tr>
<td></td>
<td>(_:b₂, :worksWith, _:b₃)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph Diagram](image-url)
### Domain-Restricted Graphs: Example

<table>
<thead>
<tr>
<th>Fang Wei</th>
<th>Stefan Woltran</th>
<th>Stefan Decker</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>$G_2$</td>
<td>$G_3$</td>
</tr>
<tr>
<td>$D_1$</td>
<td>$D_2$</td>
<td>$D_3$</td>
</tr>
<tr>
<td>$D_1 = \text{TUV} =$</td>
<td>$D_2 = \text{TUV} \cup \text{Alumni} =$</td>
<td>$D_3 = \text{DERI} =$</td>
</tr>
<tr>
<td>{:fangwei, :stefanwoltran,</td>
<td>{:fangwei, :stefanwoltran, :reinhardpichler, :thomaseiter, ... }</td>
<td>{:stefandecker, :axelpolleres, :manfredhauswirth ... }</td>
</tr>
<tr>
<td>:reinhardpichler, :thomaseiter, ... }</td>
<td>:reinhardpichler, :thomaseiter, :axelpolleres, :manfredhauswirth, ... }</td>
<td></td>
</tr>
</tbody>
</table>

- $D_1 = \text{TUV} = \{\text{:fangwei, :stefanwoltran, :reinhardpichler, :thomaseiter, ...}\}$
- $D_2 = \text{TUV} \cup \text{Alumni} = \{\text{:fangwei, :stefanwoltran, :reinhardpichler, :thomaseiter, :axelpolleres, :manfredhauswirth, ...}\}$
- $D_3 = \text{DERI} = \{\text{:stefandecker, :axelpolleres, :manfredhauswirth, ...}\}$
## Domain-Restricted Graphs: Example

<table>
<thead>
<tr>
<th>Fang Wei</th>
<th>Stefan Woltran</th>
<th>Stefan Decker</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$ $D_1$</td>
<td>$G_2$ $D_2$</td>
<td>$G_3$ $D_3$</td>
</tr>
<tr>
<td>($:_b_1$, foaf:name, &quot;Fang&quot;), ($:_b_2$, foaf:name, &quot;Stefan&quot;), ($:_b_3$, foaf:name, &quot;Reini&quot;), ($:_b_1$, worksWith, $:_b_2$), ($:_b_2$, worksWith, $:_b_3$)</td>
<td>($:_b_1$, foaf:name, &quot;Stefan&quot;), ($:_b_2$, foaf:name, &quot;Reini&quot;), ($:_b_3$, foaf:name, &quot;Fang&quot;), ($:_b_1$, worksWith, $:_b_2$), ($:_b_3$, worksWith, $:_b_1$), ($:_b_1$, worksWith, $:_b_3$), ($:_b_4$, foaf:name, &quot;Axel&quot;), ($:_b_1$, worksWith, $:_b_4$)</td>
<td>($:_b_2$, foaf:name, &quot;Stefan&quot;), ($:_b_1$, foaf:name, &quot;Axel&quot;), ($:_b_2$, worksWith, $:_b_1$)</td>
</tr>
</tbody>
</table>

$D_1 \subseteq D_2$

![Graph Diagram]

---

Fang Wei

Stefan Woltran

Stefan Decker

---

National University of Ireland, Galway

Science Foundation Ireland

Enabling networked knowledge.
Domain-Restricted Graphs: Example

\[
\begin{array}{c|c|c}
G_1 & G_2 & G_3 \\
\begin{array}{l}
(-:b_1, \text{foaf:name}, "Fang"), \\
(-:b_2, \text{foaf:name}, "Stefan"), \\
(-:b_3, \text{foaf:name}, "Reini"), \\
(-:b_1, \text{worksWith}, -:b_2), \\
(-:b_2, \text{worksWith}, -:b_3)
\end{array} & \begin{array}{l}
(-:b_1, \text{foaf:name}, "Stefan"), \\
(-:b_2, \text{foaf:name}, "Reini"), \\
(-:b_3, \text{foaf:name}, "Fang"), \\
(-:b_1, \text{worksWith}, -:b_2), \\
(-:b_3, \text{worksWith}, -:b_1)
\end{array} & \begin{array}{l}
(-:b_2, \text{foaf:name}, "Stefan"), \\
(-:b_1, \text{foaf:name}, "Axel"), \\
(-:b_2, \text{worksWith}, -:b_1)
\end{array}
\end{array}
\]

\[D_1 \subseteq D_2\]
## Domain-Restricted Graphs: Example

<table>
<thead>
<tr>
<th>Fang Wei</th>
<th>Stefan Woltran</th>
<th>Stefan Decker</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$ $D_1$</td>
<td>$G_2$ $D_2$</td>
<td>$G_3$ $D_3$</td>
</tr>
<tr>
<td>$(_b_1, \text{foaf:name}, \text{&quot;Fang&quot;})$, $(_b_2, \text{foaf:name}, \text{&quot;Fang&quot;})$, $(_b_3, \text{foaf:name}, \text{&quot;Fang&quot;})$, $(_b_1, \text{worksWith}, _b_2)$, $(_b_2, \text{worksWith}, _b_3)$</td>
<td>$(_b_1, \text{foaf:name}, \text{&quot;Stefan&quot;})$, $(_b_2, \text{foaf:name}, \text{&quot;Reini&quot;})$, $(_b_3, \text{foaf:name}, \text{&quot;Fang&quot;})$, $(_b_1, \text{worksWith}, _b_2)$, $(_b_3, \text{worksWith}, _b_1)$</td>
<td>$(_b_2, \text{foaf:name}, \text{&quot;Stefan&quot;})$, $(_b_1, \text{foaf:name}, \text{&quot;Axel&quot;})$, $(_b_2, \text{worksWith}, _b_1)$</td>
</tr>
</tbody>
</table>

$D_2 \not\subseteq D_3, D_3 \not\subseteq D_2$

![Graph Diagram](Image)
Domain-Restricted Graphs $\langle G, D \rangle$: Definition

- **Base notion in RDF semantics:** RDF interpretation for a Graph $G$

  \[ I = (Res, Prop, Lit, \epsilon, IS, IL) \]

- **We define the D-restriction of RDF interpretations:**

  \[ I_D = (Res \cap D, Prop, Lit \cap D, \epsilon, IS_{Res\cap D}, IL_{Res\cap D}) \]

- **Entailment for domain-restricted graphs,** defined as wrt. D-restriction of RDF interpretations:

  \[ \langle G_1, D_1 \rangle \models \langle G_2, D_2 \rangle \]
Domain-Restricted Graphs: Properties

- $D_1 \not\subseteq D_2$ implies $\langle G_1, D_1 \rangle \not\models \langle G_2, D_2 \rangle$
- $G_1 \models G_2$ implies $\langle G_1, D \rangle \models \langle G_2, D \rangle$

- But: Complexity of D-entailment is $\Pi^p_2$ ... Uh?

Example:
$D = \{a, b\}$

- Intuitively:
  More entailments by implicit equalities if $|D|$ is small enough!
Complexity proof (Ideas)

**Membership:** non-entailment in $\Pi^p_2$:
- We can assume w.l.o.g. that $G_1$ is ground
- “$\langle G_1, D \rangle$ does not d-entail $\langle G_2, D \rangle$” can be decided in $\Sigma^p_2$ by
  1. Guessing a $D$-interpretation such that $G_1$ is true
  2. Check that $G_2$ is false for all possible assignments of bnodes to elements of $D$

**Hardness proof by a reduction from a special variant of $H$-subsumption*, for $|D| \geq 4$ … long version.

* “total binary $H$-subsumption” i.e., no constants are allowed in clauses and only binary predicates, fixed finite Herbrand universe
Now how to remedy the mess we did…

• … we saw the first “restriction” made things more complex.

• But: *bounded treewidth* helps!
Bounded Treewidth for RDF graphs:

- Measure of “acyclicity”
- Roughly:
  “If I can decompose the graph to a tree of hyper-edges with at most $k-1$ nodes per edge, then the graph has treewidth $k$”

- Example:
Bounded Treewidth for RDF graphs:

- Measure of “acyclicity”
- Roughly:
  
  “If I can decompose the graph to a tree of hyper-edges with at most \( k - 1 \) nodes per edge, then the graph has treewidth \( k \)”

- Example:

  “Skeleton” relevant for tree-decomposition:

  \[ tw(G_2) = 2 \]
Polynomial time Algorithm for Entailment with Bounded Treewidth for $G_2$ (Idea):

- From the decomposition, process the **induced subgraphs** “bottom-up” in a modular fashion, computing partial bnode assignments.
- When going upwards, filter allowed assignments by **semi-joins** with the assignments for the child nodes.
- If an assignment “survives” at the root, entailment holds.
- $O(n^k)$ for entailment checks per node
- $O(n^{2k})$ per semi-join
- Thus, for $|G_2| = m$ we get as upper bound: $O(m^2+mn^{2k})$
Polynomial time Algorithm for Entailment with Bounded Treewidth for $G_2$ (Idea):

- From the decomposition, process the induced subgraphs “bottom-up” in a modular fashion, computing partial bnode assignments.
- When going upwards, filter allowed assignments by semi-joins with the assignments for the child nodes.
- If an assignment “survives” at the root, entailment holds.
- $O(n^k)$ for entailment checks per node
- $O(n^{2k})$ per semi-join
- Thus, for $|G_2| = m$ we get as upper bound: $O(m^2 + mn^{2k})$
Polynomial time Algorithm for Entailment with Bounded Treewidth for $G_2$ (Idea):

- From the decomposition, process the *induced subgraphs* “bottom-up” in a modular fashion, computing partial bnode assignments.
- When going upwards, filter allowed assignments by *semi-joins* with the assignments for the child nodes.
- If an assignment “survives” at the root, entailment holds.
- $O(n^k)$ for entailment checks per node
- $O(n^{2k})$ per semi-join
- Thus, for $|G_2| = m$ we get as upper bound: $O(m^2 + mn^{2k})$
Polynomial time Algorithm for Entailment with Bounded Treewidth for $G_2$ (Idea):

- From the decomposition, process the \textit{induced subgraphs} “bottom-up” in a modular fashion, computing partial bnode assignments.
- When going upwards, filter allowed assignments by semi-joins with the assignments for the child nodes.
- If an assignment “survives” at the root, entailment holds.
- $O(n^k)$ for entailment checks per node
- $O(n^{2k})$ per semi-join
- Thus, for $|G_2| = m$ we get as upper bound: $O(m^2 + mn^{2k})$
Now what about D-entailment with bounded treewidth?

• Overall complexity drops from $\Pi^p_2$ to coNP:
  Recall from above:

  - “$\langle G_1, D \rangle$ does not d-entail $\langle G_2, D \rangle$” can be decided in $\Sigma^p_2$ by
    1. Guessing a D-interpretation such that $G_1$ is true
    2. Check that $G_2$ is false for all possible assignments of bnodes to elements of $D$

• Step 2. can be done in polynomial time for bounded tree-width.

• coNP-hardness still holds (proof by 3-colorability, see paper.)
Summary:

• Some form of domain-restriction may be useful for graphs on the Web…
  – … but comes at some cost!
  – Things are not that bad unless we expect small domains (less elements than bnodes)

• Similar results for
  – enumerated classes in (fragments of) OWL?
  – entailment with finite datatypes?, etc.

  ➔ Future work!

• Bounded treewidth is more general than acyclicity. **Good news!** (if we don’t expect graphs with large cycles among bnodes)